RKAP-Tree: Efficient Multidimensional Region Searching Algorithm

Tejal M. Vandole[1] Priyanka S. Waghode[2] Yasmin B. Pinjari[3] Sayali J. Thakur[4] Ass. Prof. Minal T. Kolhe[5]

[5]Assistant Professor & HOD

[1,2,3,4,5]Department of Computer Engineering

[1,2,3,4,5]KCES's College of Engineering and Technology, K.B.C North Maharashtra University, Jalgaon, Maharashtra, India

Abstract— Recent applications like location-based-provider and social-network-based should handle continuous spatial approximate key-word queries over geo-textual streaming data of enormous amount. The optimization of continuous query processing remains an huge difficulty and performance trouble of both location information and textual information must be equivalent for every arriving streaming data tuple is more serious for spatio-textual streaming data. For geo-textual streaming data the foremost current stage in the evolution of endless spatial-keyword query approaches normally inadequacy of both supports for approximate key-word matching and excessive-performance processing query data. Progressing to address this problem, We propose a completely precise Adaptive spatial-key-word Partition structure, namely AP-Tree, to successfully organize a large variety of queries, the improvement of AP-Tree is adaptable to the spatial and key-word distributions below the guide of a cost model.. Searching is predicated on the Rabin Karp fast string matching algorithm. It's also a hash based method and is functioning on massive pool of data, for achieving higher throughput and excessive efficiency performance enhancement compared to the sooner techniques.

Keywords: Spatio-Textual Query, APTree, RK Search

I. INTRODUCTION

We investigate the matter of processing an out sized amount of continuous spatial-keyword queries over streaming data, which is vital in many applications like location-based recommendation [1] and sponsored search [2], advertising. Monitoring the geo-textual streaming data is critical to efficiently support Geographical system applications. Such as, in tweeter applications, users often subscribe some requests containing both location information and textual contents. Thus, the Geographical system applications should monitor incoming geo-textual streaming data to search out matched messages and notify the users during a period of sometime [3].

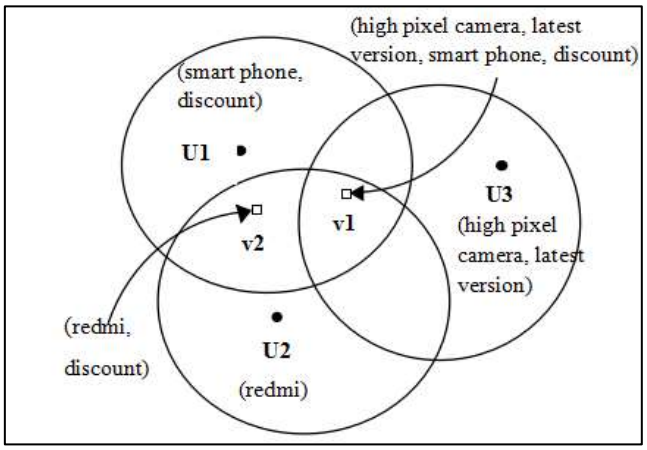

Fig. 1: Location-aware e- voucher system

1) Example 1.

Fig.1 demonstrates a location-aware publish/subscribe system which delivers e-voucher to potential consumers. A user may register her interest as a query specified by a set of keywords and a spatial region. For instance, user Ul wants to keep an eye on the discount smart phone from nearby shopping malls, and hence issues a query with keywords {smart phone, discount} and a circular region as shown in Fig. 1.

Suppose two geo-tagged e-voucher vl and v2 are released from two shops. Obviously, a e-voucher matches a query if the e-voucher's location is within the query's region, and all the query's search keywords are contained in the e-voucher Therefore, in this example, vl will be delivered to {Ul,U3} and v2 will be sent to {U2}.

The continual queries are an efficient technique for monitoring purposes over streaming data. For geo-textual streaming data, the performance issue of continuous query is more serious since both location information and textual description should be matched for each incoming streaming data tuple. The query approaches are presiding for the expansion of continuous query processing, since they're going to avoid expensive operations of index prolongation comparing to the knowledge alternatives [4]. Meanwhile, because geo-textual streaming data tuples arrive rapidly from data sources, high-performance processing could also be a key requirement for current continuous query methods. In, Xiang Wang [5] presents a correct framework consisting of knowledge types and operations needed to support geo-streaming data. In this analysis searching can be done by string matching like Rabin Karp algorithm it offers a lot of correct results as compared to approximate string matching. Thus its varied applications like bioinformatics, intrusion detection, genetic computing, plagiarism detection, digital forensics, text mining, image and video retrieval, and lots of a lot of.

String or pattern matching algorithms are accustomed notice the occurrences of a pattern during a given text or an oversized pool of strings. It examination the hash value of strings instead of the letters directly. The characteristic of the algorithm is that the hash function exploits bitwise operations and additionally considers regarding the size of the alphabet and also the length of the pattern. The algorithmic rule is use for looking multiple question strings and with the increasing variety of cores they need achieved an honest speed from string matching [6]. Additionally, this study only handled the range query amongst a spread of continuous queries since our study is based on AP-tree and also the spatial query related to AP-tree is range query.

II. RELATED WORK

This section describes the approaches for static data consider that every spatial object is stored during a spatial database and every spatial object is described with a collection of keywords. Thus, these methods indexed both the spatial information and textual keywords of every spatial object to support spatial-keyword queries. However, current indexing methods for continuous spatial-keyword queries lack the carry for approximate keyword search. Data that's stored inside the memory must be retrieved. Now, various techniques exist, which might be accustomed search and retrieve a required element from the info set. The foremost widely known algorithms used for trying to find part during a given array are Linear Search and Binary Search.

The nearest neighbor search (NNS) problem is an import research topic in image progressing, computer vision and lighting tricks. Linjia Hu [7] gives the aim of NNS is to look for the k closest points during a target set S for any point in query set Q. The keyword search for fetching approximate string matches frequently required when searching geo-textual objects, consistent with descriptions in [8].

Sahil Sharma [9] gives this proposed mechanism NLCS based string approximation method is implanted with traversing method of B-tree for searching nearby keyword stored in B-tree in order that misspelled keywords can directly used as searching keys without bypassing the indexing system. For instance, authors in [10] proposed a hybrid indexing structure that sustain classical inverted lists for infrequent document terms and further expanded R-trees for more chronic geo-textual terms. A more inclusive introduction about processing GeoStreams is available in [11]. In this paper [12], they proposed an R-tree based index structure by combining textual descriptions into R-tree nodes.

Similarly, Zhang et al. proposed an Information retrieval R-tree (IR-tree) [13] that is the fusion of R-tree and inverted files for searching geo-textual data. In specific, they proposed a hybrid index, called IQ-tree, and novel cost models for organizing a stream of incoming Boolean Range Continuous queries[14]. Any other spatial indexing structures have also been used for geo-textual data. However, KD-tree, a generalized binary tree can decrease the time complexity using spatial data structures. KD tree stores just one kind of data and it reduced height of tree. In [15] an inverted-KD tree was constructed for indexing geo-textual data.

For well distributed point sets, an efficient NNS algorithm in Majid Ahmadi [7], gives KNN tends to look the closest neighbor(s) for a target within the entire training set, hence, the prediction step of KNN is kind of time consuming. KNN could be a method for classification and regression proposed by Cover and Hart, which is one in every of the only methods in data processing classification technology [16].

In contrast to the present indexing methods for the continual spatial-keyword queries, this paper focuses on the challenges of (1) Firstly, an enormous number of queries, typically within the order of millions, are registered in many applications, and hence even a little increase in efficiency ends up in significant savings. (2) Enabling a high-performance solution to take care of the computational performance of the proposed indexing approach for streaming data. (3) Thirdly, novel techniques have to be created to develop spatial-textual indexing mechanism that adapts to both the spatial and keyword distributions of the query workload. (4) Approximate search of keywords.

III. AP TREE FRAMEWORK

AP-Tree is a f-ary tree in which queries are divided recursively in keyword or spatial partitions. To enhance the textual filtering performance, we continuously and effectively incorporate a variant of ordered keyword trie structure. Adaptiveness. Intuitively, with respect to different location and keyword distributions of the query workload, both textual feature and spatial feature may become the dominant factor. Although the keyword filtering component (e.g., local inverted list) is augmented to tree nodes, their overall performance is unavoidably deteriorated. In particular, two partition strategies are convenient, namely spatial partition and keyword partition, are proposed to repeatedly partition a set of queries by spatial feature and textual feature, respectively. A node partitioned by textual (resp. spatial) feature is called keyword (resp. spatial) node.

Notation	Definition
o	a spatial textual object
q	a continuous spatial-keyword query
o.$ (o.@)	a set of keywords for object o (query q)
o.loc (o.r)	object location (query region)
w, w_i, w_j	keyword (term)
Q (Q)	query set (subset of Q)
O (O)	object stream (subset of O)
V (V)	vocabulary (subset of V)
N	a node of AP-Tree
N_l	offset of node N
N_r	spatial region of node N
f	fanout of AP-Tree node
$@_q$	partition termination threshold
$@_{KL}$	KL-Divergence threshold
Inf	Infinity

Table 1: The Summary of Notations

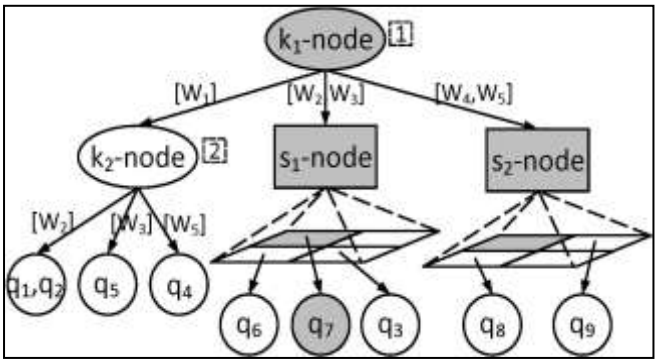

Fig. 2: AP-Tree

A. AP Tree Structure

Based on the above motivations, we devise an adaptive spatial-textual partition tree (AP-Tree for short) shown in Fig. 2 which employs keyword partition and spatial partition methods to recursively divide queries in a top-down manner. In the given structure, *N* denotes an AP-Tree node and there are three different types of nodes: keyword node (k-node), spatial node (s-node), and query node (q-node). An intermediate node is a keyword (resp. spatial) node if keyword partition (resp. spatial partition) is adopted. We use f to denote the fan out of the intermediate node. A leaf node of AP-Tree corresponds to a q-node, and each query will be assigned to one or multiple query nodes according to its query region and ordered query keywords. Below, we introduce keyword node and spatial node in details.

1) Keyword Node

We assume there's a complete order among keywords within the vocabulary *V*, and keywords in each object and query are sorted accordingly. We hold up the discussion of the effect of keyword order strategy to the experimental part Queries assigned to a node *N* are partitioned into *f* ordered cuts consistent with their N_l-th keywords, where N_l is termed the partition offset of the node *N*. We have $N_l < N^*_l$ if *N** is a descendant keyword node of N. An ordered cut is an interval of the ordered keywords, denoted as $c[w_i, w_j]$, where w_i and w_j ($w_{i <=} w_j$) are boundary keywords. For presentation simplicity, we use $c[w_i]$ to denote $c[w_i, w_j]$ if there's only single keyword within the cut.

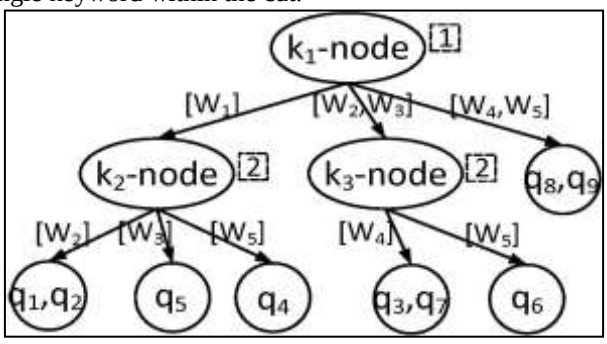

Fig. 3: Keyword Partition

2) Example 2

Fig. 3 shows a special case of AP-Tree in which only keyword partition is used on the running example. We use an oval to represent a k-node and the number on its right side indicates the partition offset. Meanwhile, a q-node is denoted by a circle. Assume there are at most 3 ordered cuts on each keyword node. In kl -node with partition offset 1; we collect the first keywords of 9 queries which correspond to $\{w_1, w_2, w_3, w_4, w_5\}$. These keywords can be divided into 3 cuts: $c[w_1]$, $c[w_2, w_3]$ and $c[w_4, w_5]$. Queries {*Ql*, *Q2*, *q4*, *q5*} are assigned to $c[w_1]$ whose corresponding node is k2-node. Since the partition offset of k2-node is 2, the second keywords of these queries, i.e., $\{w_2, w_3, w_5\}$, are used to assign queries into three cuts: $c[w_2]$, $c[w_3]$ and $c[w_5]$, each of which is associated with a q-node.

3) Spatial Node

The space is recursively partitioned by spatial nodes. Let N_r denote the region of a spatial node *N*, which is able to be divided into *f* grid cells. A query on a spatial node *N* is inserted to a grid cell *c* if *q.r* overlaps *c* or carries *c*. Note that, a spatial node may assign a query to multiple cells, unlike the keyword node within which a query is assigned to a unique cut.

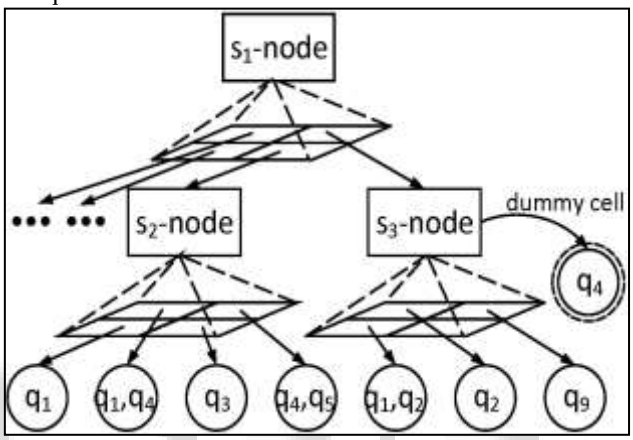

Fig. 4: Spatial partition

4) Example 3:

Fig. 4 depicts another special case of AP-Tree during which only spatial partition is utilized on the running example. We use a rectangle to represent a s-node. In each spatial node, the spatial region is partitioned into 4 cells. To matching an object, we simply navigate through the spatial nodes which contain the object location; up to we reach the leaf node. We remark that the cells on each spatial node might not be of equal size. For each keyword node *N*, a query *Q* assigned to *N* cannot find a cut if there is no enough query keywords, i.e.,$|q.\$| < N_l$. To keep these queries we use a dummy cut. Similarly, for queries each spatial node *N* include a dummy cell which contain the region of *N* (i.e., $N_r < q.r$) and hence do not need to be further partitioned on node *N*. Note that queries on the dummy cut (resp. cell) may be further partitioned by spatial (resp. keyword) node only, or simply maintained by a query node. For instance, the node indicated by the dotted circle in Fig. 4 is actually a dummy node, because the query region of *q4* fully contains the region of s_3-node.

Fig. 2 illustrates an example of AP-Tree constructed over the running example, where both keyword and spatial partitions are employed. Queries are recursively partitioned by keyword nodes or spatial nodes, and at last assigned to query nodes.

IV. AP-TREE CONSTRUCTION AND MAINTENANCE

We first propose a cost model to quantitatively analyze the goodness of keyword and spatial partitions. Then efficient

keyword and spatial partition approaches are devised to minimize the matching cost and presents the AP-Tree construction algorithm which adaptively selects keyword and spatial partition methods to construct AP-Tree in a top-down manner.

A. Cost Model

- Quantitatively measure the matching cost of keyword Partition and Spatial Partition at each node, respectively.
- *Bi* : a bucket (a Keyword cut or a spatial cell)
- *W(Bi)* : the number of query assigned to Bi
- *P(Bi)* : the hit probability that Bi is explored during the object matching
- The expected matching cost of a Partition

$$C(\mathcal{P}) = \sum_{i=1}^{f} \mathbf{w}(B_i) \times \mathbf{p}(B_i) \quad (1)$$

Where, *P* is a partition over a bundle of queries on one node. *C(P)* is the expected equivalent cost about the partition *P*. On the other hand *B* is a bucket of the partition, *w(B)* is the *B*'s weight which is the numerous queries connected to *B*, and *p(B)* is the probability that *B* is explored during the query matching. Given the object workload can be simulated by query workload, *p(B)* can be introduced as the following equation:

$$p(B) = \begin{cases} \sum_{w \in B} p(w) & if\ node\ is\ keyword\ node \\ \frac{Area(B)}{Area(N)} & if\ node\ is\ spatial\ node \\ No\ partition & else \end{cases}$$

In Equation $p(w)| = \frac{freq(w)}{\sum_{w \in P} freq(w)}$ where freq(*w*) is the frequency of keyword w among all queries in *Q*. Area(*B*) is the area of the bucket (i.e., cell) *B* and Area(*N*) is the region size of the node *N* .The optimal keyword partition of and spatial partition can be achieved by a dynamic programming algorithm and a local improvement heuristic algorithm introduced in, respectively. In addition, if *N* cannot find a cut as there are no enough query keywords then for each keyword node *N*, a query *q* is assigned to a dummy cut, i.e., $|q.\$| < N_l$. Similarly, for queries each spatial node *N* has a dummy cell whose region carries the region of *N*. Keyword Partition Without loss of generality, we assume the l-th keywords of the queries in *Q* correspond to a set of ordered keywords $V = \{w_1, w_2, \ldots, w_i\}$. Queries are partitioned into *f* ordered cuts based on their *l*-th keywords, on each keyword node, and our objective is to find an optimal keyword partition, such that the matching cost is minimized. We first present a dynamic programming approach to achieve the optimal partition.

B. Optimal Partition.

Dynamic Programming Algorithm. By Pk(i, j, c) algorithm we intend a keyword partition concerning keywords between w_i and w_j (both inclusive) with *c* cuts . The optimal partition is denoted by $P_k^*(i,j,c)$. Since keywords are sequenced, we can come up with $P_k^*(i,j,c)$ by exhausting all possible locations of the first cut as follows.

$$C(P^*_k(i,j,c)) = \min_{i<=m<=j<=c+1} (C(P^*_k(i,m,1)) + C(P^*_k(m+1,j,c-1))) \quad (2)$$

Let $P_k^*(i, m, 1)$ represent the optimal partition which consists of one cut $c[w_i, w_m]$, we have

$$C(\mathcal{P}^*_k(i,m,1)) = (\sum_{j=i}^{m} \mathbf{w}(w_j)) \times (\sum_{j=i}^{m} \mathbf{p}(w_j)) \quad (3)$$

Where w(w_j) denotes the number of queries whose l-th keyword equals w_j. 'Algorithm 1' illustrates our dynamic programming method for optimal keyword partition. In particular, Lines 1-2 compute the cost for each partition with single cut. Then Lines 3-5 iteratively compute the optimal partitions with c cuts ($2 < c < f$ -1). Finally, the optimal keyword partition P_k^*corresponds to $P_k^*(l, |V|, f)$. The time complexity of this algorithm is $O(f * |V|^2)$.

1) Algorithm 1: Optimal Keyword Partition (V, f)

Input: V: keyword set to be partitioned
f: number of cuts
Output: P_k^*: optimal *keyword partition*

```
1.  for 1<=i<=j<= |V| do
2.          Compute C(Pk*(i, j, 1)) based on Equation 3;
3.  for 2 <=c <=f —1 do
4.          for 1<=i<=|V|+1—c do
5.                  Compute C(Pk*(i,|V|,c))based on Equation 2 ;
6.  Compute C(Pk*(1,|V|,f))based on Equation 2 ;
7.  return Pk*(1,|V|,f)
```

C. Spatial Partition

Without loss of generality, we assume $f = m$ x n and *Ps* represents a spatial partition o f the node N which divides the region into m x n grid cells (buckets). We first show that it is an NP-hard problem to find optimal spatial partition.

D. Index Construction

Algorithm 2 BuildIndex(N, Q, I, *k*P, *s*P) presents the procedure of AP-Tree construction, which recursively divides queries through keyword and spatial partitions. Given a collection *Q* of queries passed from parent node, the present node *N* is also set to q-node, k-node or s-node. Particularly, there are two flags, *sP* and *kP*, which want to indicate if queries in *Q* may be further partitioned by keyword and space, respectively. Line 2 keeps all queries in an exceedingly q-node if the quantity of queries doesn't exceed a given threshold *@q* (i.e., $|Q| < @q$) or queries cannot be split further by keyword or spatial partitions (i.e., *kP* is false and *sP* is false). If keyword partition is allowed (i.e., *kP* is true), Line 6 explores keyword partition with offset *l*, and the cost is recorded by *Ck*. Recall that offset *l* indicates that the *l*-th keywords from queries in *Q* are employed for keyword partition. By *Cs* we record the price of spatial partition at Line 8 if *sP* is true. Then we are able to decide the current node *N* to be constructed from keyword partition (Line 10) or spatial partition (Line 18) base on *Ck* and *Cs*. The queries in *Q* are added to related child nodes (i.e., cuts and cells) for further processing (Line 16 and Line 24), within which the partition offset is increased by one if keyword partition is adopted. Additionally to regular cuts (cells), we also maintain dummy cut (cell) for k-node (s -

node). Specifically, we maintain a dummy cut for a k-node specified queries whose keywords are exhausted (i.e., $|q.\$| < l$) are pushed to the dummy cut with *kP* set to false (Lines 11-13). Similarly, Lines 19-21 push all queries with regions containing the node N to the dummy cell for further potential keyword partition, where the flag *sP* is about to false. Finally, the AP-Tree may be constructed by the function BuildIndex (root, Q, 1, true, true).

1) Algorithm 2: BuildIndex(N, Q, l, kP, sP)

Input: N: Current node, *Q*: a set of queries
l: keyword partition offset to be used in N
*k*P and *s*P : flags for keyword and spatial partitions
Output: AP-Tree

```
1.  if (kP is false and sP is false) or |Q|< @q then
2.          N is a q-node
3.          return
4.  Ck :=+ Inf; Cs := + Inf ;
5.  if kP is true then   /* Try Keyword Partition*/
6.          Ck ←— keyword partition on Q with offset l ;
7.  if sP is true then  /* Try Spatial Partition*/
8.          Cs ←— spatial partition  on Q ;
9.  if keyword partition is chosen (i.e., Ck< Cs) then
10.         N is a k-node with offset Nl = l;
11.         Q' ←— queries {q} in Q with |q.$|<l ;
12.         B' ←— dummy cut of N;
13.         BuildIndex(B', Q', l+1, kP = false , sP);
14.         for each child node (i.e., cut) B of node N do
15.           QB ←—queries in Q - Q' which hit the cut B;
16.           BuildIndex(B', QB, l+1, kP, sP);
17. else
18.         N is a s-node;
19.         Q' ←— queries in Q which contains Nr ;
20.         B' ←— dummy cell of N;
21.         BuildIndex(B',Q' ,l , kP , sP= false);
22.         for each child node (i.e., cell) B of node N do
23.           QB ←—queries in Q - Q' which overlap or  contain B ;
24.           BuildIndex( B, QB, l ,kP ,sP) ;
```

E. Rabin-Karp

Rabin-Karp string searching algorithm computes a numerical (hash) value for the pattern *p*, and for every *M*-character substring of text *t*. Then it compares the numerical values rather than comparing the particular symbols. If any match is found, it compares the pattern with the substring by naive approach. Else it moves to next substring of *t* to check with *p*.

1) Algorithm 3: RabinKarp

Length of pattern = M;
Hash(p) = hash value of pattern;
Hash(t) = hash value of first M letters in body of text;
Do
if (hash(p) == hash(t))
brute force comparison of pattern and selected section of text
hash(t) = hash value of next section of text, one character over
while (end of text or brute force comparison == true)

The Rabin-Karp string matching algorithm computes a hash value for the pattern, yet as for each *M*-character subsequences of text to be compared. If the hash values are same then the algorithm will calculate and analyze the hash value for next *M*-character sequence. For each text subsequence, there's just one comparison required and character matching is simply required when the hash values match.

V. INDEX MAINTENANCE

In practice, we may have to dynamically maintain an AP-Tree because of registration of latest queries and deregistration of existing queries. A straightforward strategy is that we put a replacement query into its corresponding query node supported its ordered query keywords and query region, and a question node is partitioned when its number of queries exceeds the edge $@_q$. Similarly, we remove a question from its corresponding query nodes if it's deregistered and a keyword node or spatial node turns to a question node if the quantity of its descendant queries is a smaller amount than $@_q$. This approach is efficient and works well if the underlying query workload remains stable. On the downside, the partitions of the present nodes can't be adjusted to the change of query workload, and hence the performance is also deteriorated. To alleviate this issue, we adopt the well-known *KL*-Divergence [3 1] to detect the changes of underlying query workload for nodes with a selected amount of queries. Specifically, let W_{old} (*Bi*) denote the burden of the bucket Bi when the node is built while *W*(*Bi*) is calculated for all current queries. Let D_{KL} ($w_{old}|w$) denotes *KL*-Divergence of the query workload, and AP-Tree nodes are re-constructed if D_{KL} ($w_{old}|w$) exceeds a given threshold $@_{KL}$.

A Problem Formulation. We first define the notations of a geo-textual data stream. Then, we introduce the definition of the continual spatial approximate keyword query over the geo-textual data stream. Finally, the matter is stated. We remark that calculation of *KL*-Divergence value is sort of cost-free because they will be easily updated when the node is visited during the query updates. Moreover, only descendant queries of the node are involved within the re-construction. During this way, our empirical study shows that AP-Tree is self-adjustable to the workload changes with an honest maintenance overhead.

VI. RESULT AND DISCUSSION

We first provided an experimental setup, then evaluated and discussed the performances of RKAP-Tree against geo-textual streaming data. The data sets used in this paper is synthetic data. Synthetic data contains id, time (t1, t2), Keyword, coordinates (x1, y1, x2, y2). The dataset contains various sizes of data files from 500 up to 1000000 and more.

A. Evaluating and Discussing AP-Tree

In this section, we evaluate creation time, searching time, and memory size required by various algorithms along with AP-tree over a geo-textual data stream.For comparison, we

used Cell, KD Tree, and Cell-KD Tree due to the reason that in Cell based searching overlapping occurs because cells are limited and data lies on boundary is lost, KD Tree allows only one type of data and reduces height of tree, Cell-KD uses advantage of Cell and KD Tree respectively. We directly employ the specified data structures to process continuous spatial approximate keyword queries in our setting.

B. Creation Time

It is observed that AP Tree consumes less time for creation of any size of data.

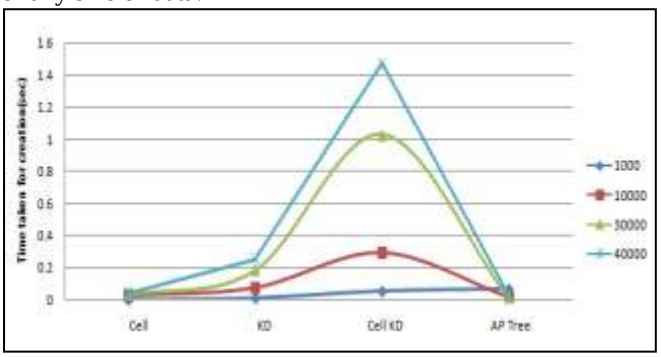

Fig. 5: Data structure creation time

C. Searching Time

In below table, shown time taken for searching for different trees like Cell, KD, Cell KD and APTree for same searching algorithm and it is clearly shown that time taken for searching in APTree is extremely less as compare to other data structures. Different colour lines indicate a searching different size of record files like 1000, 10000, 30000, 40000. And it also clear that time taken for searching different size of record in same APTree is approximately same or there is slightly difference between them.

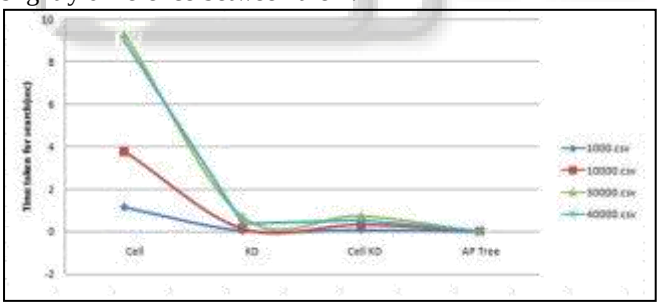

Fig. 6: Searching result of RK search

Above graph shows the searching time required to search same size of data using same searching algorithm (RK Search) on different data structures to show AP Tree produce result in less time as compare to other specified data structures.

D. Memory Size

In below table, shown memory required for different trees like Cell, KD, Cell KD and APTree, It is observed that AP Tree required more memory than other data structures due to in Cell boundary data is lost and KD Tree create two different trees to store keyword and spatial data, Cell-KD Tree combine both concepts of Cell, KD Tree. AP Tree stores Keyword data, Spatial data, and Metadata. Using metadata AP Tree reduces searching time by checking query metadata stored in nodes. Metadata in node contains range value of its child nodes. If node doesn't contain query range then it check in its sibling node and continue further.

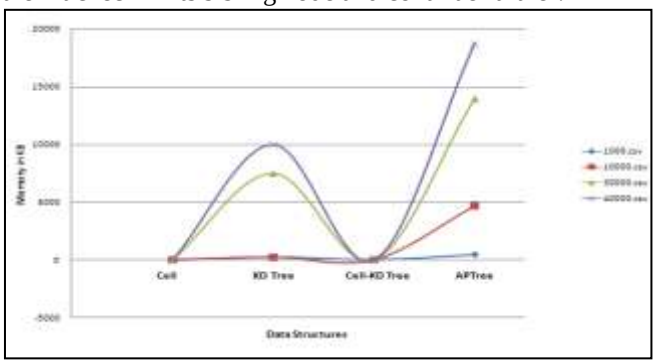

Fig. 7: Total memory required

VII. CONCLUSION

We will propose a novel adaptive spatial textual partition indexing structure, namely AP-Tree, to efficiently organize an enormous number of continuous spatial keyword queries specified each incoming object from spatial textual data are often rapidly delivered to relevant queries. Unlike the previous spatial-textual indexes which prefer either textual feature or spatial feature, AP-Tree is constructed in an adaptive way by carefully choosing keyword or spatial partitions guided by a cost model. Substantial experiments illustrates that our technique achieves a high throughput performance over streaming spatial-textual data. It is a basic requirement for several current applications to support continuous spatial-keyword queries. However, existing spatial-keyword query indexing approaches generally don't apply for approximate keyword matching. To deal with this problem, this paper first proposes an indexing approach for efficient supporting of continuous spatial approximate keyword queries by integrating RK search to AP-tree to support the approximate keyword matching between queries and streaming data tuples, namely RKAP-Tree. To improve the performance of RKAP-Tree, it will be implemented on CUDA processor.

REFERENCES

[1] M.-H. Park, J.-H. Hong, and S.-B. Cho, "Location-based recommendation system using bayesian users preference model in mobile devices" *in Ubiquitous Intelligence and Computing, Springer, 2007.*

[2] Konig, K. Church, and M. Markov, "A data structure for sponsored search", *in ICDE, 2009, pp. 90-101.*

[3] Chen, L.; Cong, G.; Cao, X." An efficient query indexing mechanism for filtering geo-textual data." *In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data,New York, NY, USA, 22–27 June 2013; pp. 749–760. Park, J.; Hong, B.; Ban, C. A continuous query index for processing queries on RFID data stream.*

[4] In Proceedings of the 13th IEEE International Conference on"Embedded and Real-Time Computing Systems and Applications", *Daegu, Korea, 21–24 August 2007; pp. 138–145.*

[5] Xiang Wang, Ying Zhang, Wenjie Zhan, Xuemin Lin, Wei Wang "AP-Tree: Efficiently Support Continuous Spatial-Keyword Queries Over Stream",*ICDE Conference 2015 IEEE.*
[6] Nayomi Dayarathne and Roshan Ragel,"Accelerating Rabin Karp on a Graphics Processing Unit (GPU) using Compute Unified Device Architecture (CUDA)", *2014 IEEE.*
[7] Linjia Hu, and Saeid Nooshabadi, Majid Ahmadi "Massively Parallel KD-tree Construction and Nearest Neighbor Search Algorithms", *2015 IEEE International symposium on circuit and systems.*
[8] Li, F.; Yao, B.; Tang, M.; Hadjieleftheriou, M. "Spatial approximate string search*", IEEE Trans. Knowle. Data Eng. 2013, 25, 1394–1409.*
[9] Sahil Sharma, Mayank Sharma, Rachna Jain, Sunil Kumar Khatri "NLCS based string approximation for searching indexing keywords in B-tree" *2017 2nd International Conference on Telecommunication and Networks (TEL-NET).*
[10] Go¨bel, R.; Henrich, A.; Niemann, R.; Blank, D. "A hybrid index structure for geo-textual searches" *In Proceedings of the 18th ACM conference on information and knowledge management (CIKM) Hong Kong, China, 2–6 November 2009; pp. 1625–1628.*
[11] Brandt, T.; Grawunder, M. Geostreams:," A survey" *ACM Comput. Surv. 2018, 51, 1–37*
[12] Guoliang Li, Yang Wang, Ting Wang, Jianhua Feng, "Location-Aware Publish/Subscribe" *KDD'13, August 11–14, 2013, Chicago, Illinois, USA.*
[13] Zhang, D.; Chee, Y.M.; Mondal, A.; Tung, A.K.H.; Kitsuregawa, M. "Keyword search in spatial databases: Towards searching by document*.", In Proceedings of the 2009 IEEE 25th International Conference on Data Engineering (ICDE), Shanghai, China, 29 March–2 April 2009; pp. 688–699.*
[14] Lisi Chen, Gao Cong, and Xin Cao, "An Effificient Query Indexing Mechanism for Filtering Geo-Textual Data " *SIGMOD'13, June 22–27, 2013, New York, New York, USA.*
[15] Li, J.; Wang, H.; Li, J.; Gao, H. "Skyline for geo-textual data. Geoinformatica ", *2016, 20, 453–469.*
[16] Wenfeng Hou , Daiwei Li, Chao Xu, Haiqing Zhang , Tianrui Li "An Advanced k Nearest Neighbor Classification Algorithm Based on KD-tree" *2018 IEEE International Conference of Safety Produce Informatization (IICSPI).*

