Database Summarization and Publishing in Wireless Environments

Anshul Gandhi and R.K. Ghosh

Indian Institute of Technology, Kanpur - 208 016, India
{ganshul, rkg}@cse.iitk.ac.in

Abstract. Data dissemination in a mobile computing environment typically uses push based data delivery model. Processing queries under this scenario is challenging because we need to organize the broadcast data to efficiently process queries of an average mobile client. In this paper, we adopt the learning technique from [2] in order to learn the patterns of queries of the average mobile user. We then propose a method to create various summary databases from the main database available at server side, on the basis of these query patterns.

1 Introduction

In this paper we view query processing as the fundamental element for data requirements in broadcast channel from an average client's prospective. A client will be interested in some data items for which he/she would like to place a demand. So in trying to learn the data demands of an average client, the approach should be to analyze the pattern of queries originating from client devices. These patterns can be stored in a compressed trie like data structure much like the ones used in location prediction of mobile devices using subscriber mobility patterns [1, 2]. We, therefore, adopt the above technique for learning query patterns. We then form and analyse the Entity-Relationship graph (E-R graph) related to these queries by extracting the database attributes from them. Next we find a spanning subgraph of the E-R graph which can be used to construct the summary database by selecting appropriate database attributes and corresponding relationships from the main database at the server side. The server then intermixes summary data with an index before pushing it on broadcast channel which facilitates energy efficient retrieval by the mobile hosts.

2 Current Prediction Techniques

Two techniques, namely, LeZi update [1] and active LeZi [2] both based on Lempel-Zvi data compression algorithm [3] have been used in prediction of the location of a mobile node by learning its mobility pattern. We model the query prediction as an instance for the Active LeZi predictor with almost no overheads. The readers interested to know more about active LeZi may refer to the original paper [2]. In the database scenario, we keep track of queries received from various clients and record these in the trie.

Let the incoming queries be $q_1, q_2, \ldots, q_k$ at some time t. Every query makes references to a certain set of attributes from a database. We assume that the required set

S. Chaudhuri et al. (Eds.): ICDCN 2006, LNCS 4308, pp. 342–348, 2006.

of attributes can be extracted from the query. Once this set is available, every query will be identified with its attribute set and this distinct set of such attributes will be relabeled as q_1, q_2, ..., etc. This solves the issue of converting a query into a single element as required by Active LeZi. We then apply the Active LeZi technique to the input $Q = \{q_1, q_2, \ldots, q_k\}$, with q_i , $i \in \{1, 2, \ldots, k\}$, belonging to a domain of elements D. Note that here D is simply the powerset of all attributes in the given database. By looking at the output probability values from the Active LeZi scheme, we can pick the query with the highest probability and use this as our prediction for the next query (attribute-wise).

Despite its ease of implementation and simplicity, Active LeZi have some drawbacks when used as a query predictor which are revealed when we take a closer look at a query sequence. There is no concept of *aging* in active LeZi. Aging captures spatio-temporal locality. As the queries keep coming, they change trends. In other words, the query patterns change with time and/or location. If aging is not included, such patterns can never be captured.

3 Augmented Active LeZi

At the client side, there are two pieces of information that we plan to make use of, namely the location of the user and time at which the query has been sent. The client simply uses mobile device id or mobile cell id for the location. For the timestamp, a standardized clock for that region can be used. Hence, without much overhead in terms of uplink bandwidth, this extra information can be obtained from the client. We now have to change our trie accordingly.

From the Active LeZi, we see that the nodes simply contain the frequency of that particular pattern. We now change this node to include a pointer to a linked-list. Each node of this linked-list would be a 3-tuple with the following components: (i) the cell node id or the zone id, (ii) the date, and (iii) the time. No particular sorting within the linked list is required. Hence, as a new query comes in, we simply increment the frequency in the corresponding trie node and create a new node in the attached linked-list with information from the client. The trie would now look like the one shown in figure 1 (a).

Since aging is to be used for improving predictions, there is no need to run a check on aging at the server unless there is a need to predict; and also, we only need to check aging for those nodes of the trie that get affected by the input query. Consider a trie at time t; and suppose that we are interested in finding the probability of the next query being q_i. Then simply check aging for all those nodes in the trie that participate in computation of the cumulative probability value for q_i. We must, however, remove the aged entries before using the node for calculating the probability for q_i.

In the Active LeZi, we make use of all orders to calculate the probability for a prediction. We use the frequencies in the trie to get the final value. Using our linked list, we can calculate the average value of the time at which that particular node is accessed. Once we have the average value, we can compare it with the value of the current time and based on the deviation between the two, we can either add or subtract a factor from the total probability value. One can also choose to multiply the deviation value with

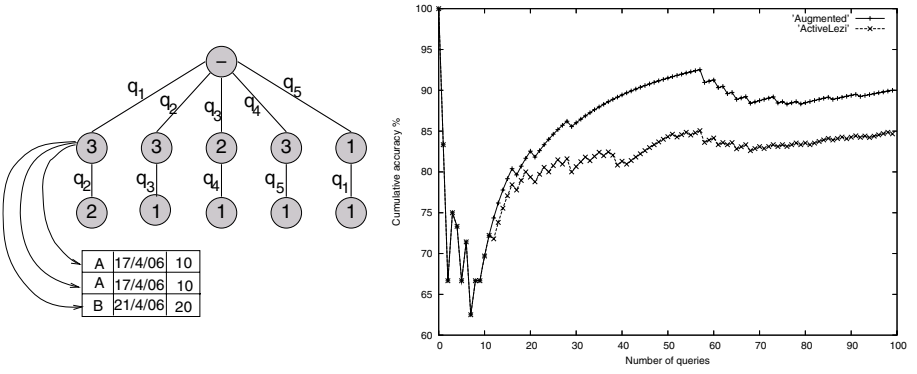

Fig. 1. Trie resulting from Augmented Active LeZi and comparison of Augmented Active LeZi with Active LeZi

the probability obtained from Active LeZi if the time factor is very significant for the database usage. In general, we can say that the probability function in the Augmented Active LeZi would be a function of the probability function used in the Active LeZi and the deviation between the average and observed values.

We tested augmented active LeZi for a simulated input data and compared its performance against active LeZi. Our test data was a set of 100 queries. From figure 1 (b), it is clear that the augmented predictor, which uses the hour of arrival of query to improve its prediction, outperforms the active LeZi by around 5% in our simulation. Since Active LeZi itself registers a success rate in the range of 80% in our case, the apparently small 5% increase in accuracy of prediction is indeed very significant. Though this simluation is in no way conclusive, it does show that there are cases where the Augmented Active LeZi will outdo the Active LeZi, especially if the incoming pattern tends to change its trends over time.

4 Constructing Summary Databases

4.1 Density of Prediction and Database Hierarchies

With the probability values at hand, we can now decide to use the highest-probability query to construct our database or we could decide to include more than one query into our summary database. We shall denote this number of queries by the parameter *density*. Note that higher the density, the larger would be the size of our database, which is not a very favourable requirement. But as the density goes up, so is the chance that the incoming query shall be answered by using the information contained in the summary database.

A prediction process is associated with an inherent chance of failure. In our case, a failure is exhibited when the client's query is not answered by the summary database. If the failure has resulted because of the low density value, then we can create another summary database which would now contain lower valued predictions which ccould answer the incoming query.

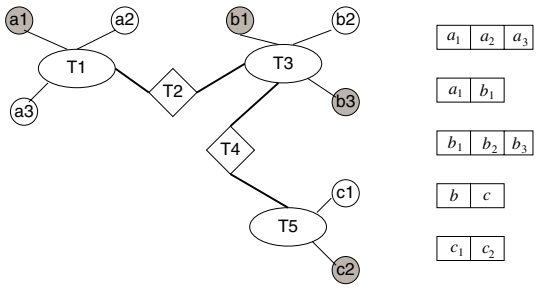

Fig. 2. E-R diagram corresponding to the example in section 4.2 and its tabular form

Assume that the predicted queries are sorted by decreasing cumulative probability values as $P = \{p_1, p_2, \ldots, p_k\}$ such that the set P contains all possible predictions (including the ones with zero probability). Say density = 3. Then, our summary database would answer queries p_1, p_2 and p_3. Let us assume that the incoming query is not in exact match with the predicted pattern and it corresponds to p_4. What we can do to avoid the loss of query is to create a second level of summary database which would have the power to answer queries p_4, p_5 and p_6. Note however that this level is at a lower priority than the first summary database. To tackle this issue, we could publish this level less number of times than compared to level 1. In fact, no matter how many such levels we have, we can publish them with a frequency which would be in a reverse ratio of their priorities.

4.2 Using E-R Modeling

We wish to ensure that we consider only those queries that are valid and reject those which are not. We also wish to do this as early as possible before updating the trie with the query. By using the E-R model corresponding to the database, we can model this issue in a different manner. Consider the E-R model shown in figure 2 and the corresponding tabular form of an arbitrary database which appears alongside.

Let the required attributes corresponding to an incoming query be the ones that are shaded in figure 2, namely, a_1, b_1, b_3 and c_2. Call these attributes as the set *S*. To ensure validity, all that is needed is to check whether there is a *marked path* between the required edges. A 'marked path' is a path that includes only those edges that are marked. Note that the E-R model is treated implicitly as a graph where the entities, relations and attributes are nodes and the edges are the edges of the graph. The outline of a simple algorithm to mark the edges is provided below.

```
 i. For all selected attributes, mark the edge connecting them to their parent
    entities.
ii. For an edge connecting a relation R to an entity E, mark it if the
    attributes of E included in R belong to S.
```

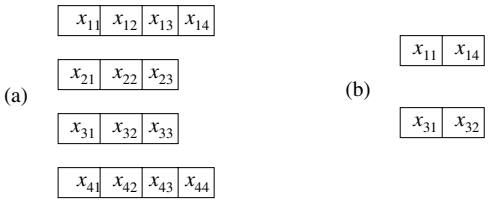

Fig. 3. Example

As an example consider the execution of above algorithm over the E-R graph shown in figure 2. After step 1, edges 1, 6, 8 and 12 gets marked in figure 2. Regarding step 2, we look at the attributes in the tabular form of R. Since R is connected to E, it must have attributes of E in its tabular form. If all these attributes belong to S, then the edge connecting R to E gets marked. So, after executing of step 2 over E-R graph of figure 2 edges 4, 5, 9 and 10 get marked. We, therefore, find that we have a marked path {1, 4, 5, 6, 8, 9, 10, 12} which touches all the attributes in S.

We can use the following steps to ensure validity of the query once the marking is done.

```
 i. Remove all unmarked edges.
ii. Check whether we have a connected component of the remaining graph that
    connects all the nodes in S.
iii. If yes, return VALID; else return INVALID.
```

4.3 Construction Steps

After ensuring the validity of the query, our next task is to decode the queries back into their attribute requirements for the database construction. We shall collect all these attributes for each selected query into a set S. We now scan all the tables of the database and mark all those attributes of the database which belong to S. The algorithm is provide below.

```
initialize S:= null;
initialize D:= input database;
for (i ≤ 0, i < density, i++) do
   for each attribute x required by p(i) do
      S := SU{x}
   endfor
endfor
for each table T in database do
   for each attribute y of table T do
      if y belong to S mark y
      endif
   endfor
endfor
remove all unmarked components from D
for each table T in D do
   for every table T′ in D such that (T∩T′ = null)
return D.
```

At this point, just by looking at the marked entries of the database, we can see what the desired output database would be like. It would be a collection of sub-tables of the original database. That is, if we delete all the unmarked attributes, then the remainder of the database would be our summary database.

Consider the following tabular schema in figure 3(a) for an input database. Here, the x's are attribute names (not necessarily distinct). Say our predictor marks attributes x_{11}, x_{14}, x_{31} and x_{32}. Our output would then be as shown in figure 3(b). Further, say x_{31} and x_{14} are actually from the same domain. That is, they share the same attribute name. We shall use the normal convention that if the attribute names are the same in a database, then these attributes belong to the same domain. We shall now save space by joining these two columns by using a *join* function call. We can simply modify the existing variants of the `JOIN` call to serve our purpose.

We require our `JOIN` to join the two tables that share any common attribute. The first common attribute found is selected and the `JOIN` is performed. Say, x_{31} is a subset of x_{14}. Then we would like to join T1 and T3 in such a way that no tuples are lost. We can use null values for the empty components of the tuples. Also, if x_{31} and x_{14} are disjoint, then we would simply have the output as one single table that would have as many tuples as there are in T1 and T3 combined and each tuple would be a 3-tuple in this case. One for x_{11}, one for x_{14}/x_{31} and one for x_{32}. In most cases, we would be saving considerable space by this method. However in cases where disjoint columns are operated on using `JOIN`, no space saving is achieved.

The extended algorithm including the `JOIN` appears below.

```
remove all unmarked components from D;
for each table T in D do
   for every table T′ in D such that (T∩T′=null) do
      join T and T′; delete T′
   endfor
endfor
return D;
```

The resulting database for the previous case would be: T1 | x_{11} | x_{14}/x_{31} | x_{32} |

5 Conclusions

In this paper, we developed an idea to connect query prediction with the active LeZi predictor. We then proposed some augmentations to the active LeZi which has added advantages of aging and localizing the temporal behaviour of queries. We then gave a construction of summary databases for the mobile environment. Summary databases can be used to minimize the amount of data transfers over the network. As indicated at the beginning, the motivation behind this work was to reduce the size of data transmission on the broadcast channel. By reducing the size of data transmission we not only achieve better utilization bandwidth but also a substantial reduction in time for data transmission over wireless broadcast channel.

References

1. BHATTACHARYA, A., AND DAS, S. K. Lezi-update: An information-theoretic framework for personal mobility tracking in pcs networks. *Wireless Networks 8* (2002), 121–135.
2. GOPALRATNAM, K., AND COOK, D. J. Active lezi: An incremental parsing algorithm for sequential prediction. In *Proceedings of the Florida Artificial Intelligence Research Symposium* (2003).
3. ZIV, J., AND LEMPEL, A. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory 24*, 5 (1978), 530–536.