Scripts for Sensor Network Seminar – Data Management Section

Lectured by George Kollios, Scribed by Feifei Li

Boston University
Computer Science Department
{gkollios,lifeifei}@cs.bu.edu

Abstract

In this section of the seminar, our focus is on the data management aspect of sensor network. We view the sensor network as a large distributed database system, namely sensor database. Recent development of sensor database systems has attracted more and more interests in the querying performance for sensor network. Most of sensor network systems involve monitoring answers to continuous queries over data streams produced at physically distributed locations, and most previous approaches require streams to be transimitted to a single location for centralized processing. Unfortunately, the continual transimission of a large number of rapid data streams to a central location can be impractical or expensive. TinyDB, COUGAR allow users to extract useful information from a sensor network using aggregation queries. These systems use in-network aggregation to reduce trasimission cost, hence reduce the energy consumptions of the network. Another interesting issue is how to make the sensor database systems be more fault-tolerant. We discuss a paper using sketches to enable duplicate-insensitive multi-path broadcasting which has good performance when there are failures within the network. We also view the sensor network from the stream database point of view where we discuss how to perform approximate join over data streams. Finally, We discussed query processing in IrisNET, which essentially answers the queries in wide-area sensor databases.

See the reference [1], [2], [3], [4], [5], [6], [7], [8]

1 TinyDB

TinyDB is a sensor database system developed at Berkeley for the project called TinyOS. The contribution of TinyDB is the design of an acquisitional query processor for data collection in sensor networks. They use in-network aggregation and are able to significantly reduce power consumption over traditional passive systems. Simple extensions to SQL has been done for controlling data acquisition, and they show how acquisitional issues influence query optimization, dissemination, and execution. For example, in the TAG(TinyDB) system, there is a base station directly connected to a sensor designated as the root node. Aggregate queries over the sensor data are formulated using a simple SQL-like language, and then distributed across the network, e.g. by smart flooding. As the query is distributed across the network, a spanning tree is formed for the sensors to return data back to the root node. At each node in the tree, the sensor combines its own values with the data received from its children, and sends the aggregate to itsparent. TinyDB performs reordering on the query predicate to optimize the query process. They also propose other ways of optimizing query execution plan for sensor database. If there are no failures, this technique works extremely well for decomposable aggregates, namely distributive and algebraic aggregates such as MIN, MAX, COUNT and AVG. TAG papers categorize the aggregates query into four dimensions:

- Duplicate Sensitive,
  Max Min are not duplicate sensitive, Sum and Average are duplicate sensitive.
- Exemplary or Summary,
  Max, Min are exemplary, Count and Sum are Summary.
- Monotonic,
  Max Min Sum Count are monotonic, Average is not monotonic.

- Size of Partial State,
  this classifies the aggregate based on the size of its partial state.

TindyDB(TAG) supports event-based query and periodic query. This system does not perform well when there are node failures or link failures in the network. Significant amount of information in these cases will be lost, and hence generate wrong aggregated result at the base station.

2 Approximate Aggregation using Sketches

To improve the performance of in-network aggregation queries and make it more fault-tolerant, we discuss a robust and scalable method for computing duplicate sensitive aggregates. Since exact solutions are generally impractical to guarantee in the face of losses in sensor database, we provide an approximate solution which is robust against both link and node failures. The idea can be summarized as follows:

- First, We extend well-known duplicate insensitive Flajolet and Martin sketch to support SUM aggregates.
- Then We combine duplicate insensitive sketches with multi-path routing techniques to produce highly accurate sketches with low communication and computation overhead.

The FM Count sketch is defined as:

Definition 1 *Given a multi-set of items* $M = \{x_1, x_2, x_3, \ldots\}$*, the* distinct counting *problem is to compute* $n \equiv |\mathrm{distinct}(M)|$.

Given a multi-set M, the FM sketch of M, is a bitmap of length k. The entries of bitmaps are initialized to zero and are set to one using a random binary hash function h applied to the elements of M. Formally,

$$S(M)[i] \equiv 1 \text{ iff } \exists x \in M \text{ s.t. } \min\{j \mid h(x,j) = 1\} = i.$$

By this definition, each item x is capable of setting a single bit in $S(M)$ to one – the minimum i for which $h(x,i) = 1$. It is proven that this will give a good approximation of the distinct count of N, but with a relative large variance. To improve the accuracy and variance, we could use a larger k and multiple bitmaps (insert into each bitmap independently).

In sensor database, we still have to handle for Sum, idea is to simulate Sum in FM sketch using Count. The distinct Sum problem is defined as:

Definition 2 *Given a multi-set of items* $M = \{x_1, x_2, x_3, \ldots\}$ *where* $x_i = (k_i, c_i)$ *and* c_i *is a non-negative integer, the* distinct summation *problem is to calculate*

$$n \equiv \sum_{\mathrm{distinct}((k_i,c_i) \in M)} c_i.$$

The algorithm for distinct sum is shown here (get from the paper):

Algorithm 1 SUMMATIONINSERT(S,x,c)

```
1:  d = pick_threshold(c);
2:  for i = 0, ..., d - 1 do
3:      S[i] = 1;
4:  end for
5:  a = pick_binomial(seed=(x, c), c, 1/2^d);
6:  for i = 1, ..., a do
7:      j = d;
8:      while hash(x,c,i,j) = 0 do
9:          j = j + 1;
10:     end while
11:     S[j] = 1;
12: end for
```

The basic intuition: set the bits in the summation sketch *as if* we had performed c_i successive insertions to an FM sketch. The method proceeds in two steps: we first set a prefix of the summation sketch bits to all ones, and then set the remaining bits by randomly sampling from the distribution of settings that the FM sketch would have used to set those bits.

3 Approximate Join Over Data Stream

In this problem, we discuss how to perform approximate join query over two data streams. The problem is aroused as the limitation of processing power in online computation of data streams and also the high arrival rate of the data streams. Basic idea is to use Sliding Window frames to perform approximate joins using limited memory space. In the paper we presented, they present an optimal solution for fast-cpu offline scenario, where the problem is modelled as finding the min-cost flow-path within the network. They then propose using a PROB heuristic to decide which tuple to drop for fast-cpu online case. The PROB heuristic is defined as:

Definition 3 *Given two streams $R = \{r_1, r_2, r_3, \ldots\}$ $S = \{s_1, s_2, s_3, \ldots\}$ where r_i and s_i is a tuple item in stream R, S at time instance i respectively. We also have a statistics as $Pr(r_i)$ = probability of tuple r_i having matching tuple in stream S from time $i \leq t \leq \infty$. The* PROB Heuristic *is simply defined as drop the tuple with lower probability, and solving ties by keeping the tuple that newly arrives.*

They show in experiments that the PROB Heuristic performs almost as good as the fast-cpu offline algorithm which gives a upper-bound on any online algorithm. PROB Heuristic is getting better with the increase of skew in the data stream. But when the data stream is uniformly distributed it is as good as Random Load Shedding.

In general, carefully designed Semantic Load Shedding will be better than Random Load Shedding, but there are still some open problems remained unsolved for this topic:

1. How to generalize the heuristic for multiple streams, not just two?
2. What about the correlations in the data streams? Is that going to affect the performance or we could design better algorithms when taking that into account?
3. How to combe life time of tuple into the heuristic?
4. Will it be good if we could design a heuristic that take the temporal distribution of tuples in the stream into consideration? This is the idea of caching, and reference locality.

4 IrisNET:Query in Wide Area Sensor Database

In this topic we focus on querying wide area sensor databases, containing (XML) data derived from sensors spread over tens to thousands of miles. IrisNET is a scalable system for executing XPATH queries on such databases. IrisNET maintains the logical view of the data as a single XML document, while physically the data is fragmented across any number of host nodes. For scalability, sensor data is stored close to the sensors, but can be cached elsewhere as dictated by the queries. IrisNET tries to enable self-starting distributed queries that jump directly the lowest common ancestor of the query result. IrisNET tries to increase query throughputs and decrease query response times in wide area sensor databases. The basic idea is:

- Using XPATH queries on XML databases
- For high update and query throughputs, the sensor database is partitioned across multiple sites operating in parallel, and the queries are directed to the sites containing the answers
- A DNS alike server is designed to direct the queries to the different level in the hierarchical database

References

[1] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate aggregation techniques for sensor databases. In *ICDE*, 2004.

[2] A. Das, J. E. Gehrke, and M. Riedewald. Approximate join processing over data streams. In *SIGMOD*, 2003.

[3] A. Deshpande, S. Nath, P. Gibbons, and S. Seshan. Cache-and-query for wide area sensor databases. In *SIGMOD*, 2003.

[4] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31, 1985.

[5] S. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. In *OSDI*, 2002.

[6] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W.Hong. The design of an acquisitional query processor for sensor networks. In *SIGMOD*, 2003.

[7] Y. Yao and J. Gehrke. The cougar approach to in-network query processing in sensor networks. *SIGMOD Record 31(3)*, 2002.

[8] Y. Yao and J. Gehrke. Query processing in sensor networks. In *CIDR 2003*, 2003.