Robust Aggregation in Sensor Network: An Efficient Frequent itemset and Number of occurrence counting

Kayalvizhi S \(^{(1)}\) and Vanitha k \(^{(2)}\)

\(^{1}\) M.Phil Scholar, SITS, Dr. G R D College of Science, Coimbatore.

\(^{2}\) Assistant Professor, SITS, Dr. G R D College of Science, Coimbatore.

Abstract

Sensor networks are collection of sensor nodes which co-operatively send sensed data to base station. As sensor nodes are battery driven, an efficient utilization of power is essential in order to use networks for long duration hence it is needed to reduce data traffic inside sensor networks, reduce amount of data that need to send to base station. The aim of the project is to develop scalable aggregation methods to extract useful information from the data the sensors collect. Partitioning large set of data, for the result of horizontal aggregation, in to homogeneous dataset is important task in this system. Association rule apriority algorithm using SQL is best suited for implementing this operation. In this project we consider the PIVOT operator which is a built-in operator in a commercial DBMS. Since this operator can perform transposition it can help evaluating horizontal aggregations, our proposal though the list of distinct to values must be provided by the user, whereas ours does it automatically, output columns are automatically created.

Index Terms- Data aggregation, Wireless Sensor Network (WSN), SPJ, Horizontal aggregation, case, pivot, Frequent itemset.

1. Introduction

Wireless Sensor Network (WSN) consists of spatially distributed autonomous sensors to cooperatively monitor physical or environmental conditions, such as temperature, sound, vibration, pressure, motion or pollutants. That’s why most of the research on WSNs has concentrated on the design of energy and computationally efficient algorithms and protocols, and the application domain has been confined to simple data-oriented monitoring and reporting applications. WSNs nodes are battery powered which are deployed to perform a specific task for a long period of time, even years \([1, 2]\). Sensor nodes usually generate significant redundant data. So, to reduce the number of transmission, similar packets from multiple nodes can be aggregated. Data aggregation is the combination of data from different sources according to a certain aggregation function, e.g., duplicate suppression, minima, maxima and average. It is incorporated in routing protocols to reduce the amount of data coming from various sources and thus to achieve energy efficiency.

Figure 1 – data aggregation process

In a typical sensor network, each sensor produces a stream of sensory observations across one or more sensing modalities. But for many applications and sensing modalities, such as reporting temperature readings, it is unnecessary for each sensor to report its entire data stream in full fidelity. Moreover, in a resource-constrained sensor network environment, each message transmission is a significant, energy-expending operation. For this reason, and because
individual readings may be noisy or unavailable, it is natural to use data aggregation to summarize information collected by sensors. As a reflection of this, a database approach to managing data collected on sensor networks has been advocated\cite{9,10}, with particular attention paid to efficient query processing for aggregation queries.

In the TAG system, users connect to the sensor network using a workstation or base station directly connected to a sensor designated as the sink. Aggregate queries over the sensor data are formulated using a simple SQL-like language, then distributed across the network. Aggregate results are sent back to the workstation over a spanning tree, with each sensor combining its own data with results receive from its children. If there are no failures, this in-network aggregation technique is both effective and energy-efficient for distributive and algebraic aggregates such as MIN, MAX, COUNT and AVG \cite{5}. However, as we will argue, this technique is much less effective in sensor network scenarios with moderate node and link failure rates. Node failure is inevitable when inexpensive, faulty components are placed in a variety of uncontrolled or even hostile environments. Similarly, link failures and packet losses are common across wireless channels because of environmental interference, packet collisions, and low signal-to-noise ratios \cite{8}. Retransmission-based approaches are expensive in this environment, so solutions based upon multi-path routing were proposed in. For aggregates such as MIN and MAX which are monotonic and Exemplary, this provides a fault-tolerant solution. But for duplicate-sensitive aggregates such as COUNT or AVG that give incorrect results when the same value is counted multiple times existing methods are not satisfactory.

We propose a new class of aggregate functions that aggregate numeric expressions and transpose results to produce a data set with a horizontal layout. Functions belonging to this class are called horizontal aggregations. Horizontal aggregations represent an extended form of traditional SQL aggregations \cite{4}, which return a set of values in a horizontal layout (somewhat similar to a multidimentional vector), instead of a single value per row. This article explains how to evaluate and optimize horizontal aggregations generating standard SQL code.

2. Existing System

In this paper we have used the existing approaches to give more weight to our work. The below mentioned approaches are stated as under:

1. In-Network Aggregate Query Processing.
2. Best-Effort Routing in Sensor Networks.
3. Counting Sketches.

2.1 In-Network Aggregate Query Processing:

A simple approach to evaluate an aggregation query is to route all sensed values to the base station and compute the aggregate there. Although this approach is simple, the number of messages and the power consumption can be large. A better approach is to leverage the computational power of the sensor devices and compute aggregates in-network \cite{9}. Aggregates that can be computed in-network include all decomposable functions.

The in-network query evaluation has two phases, the distribution phase and the collection phase. During the distribution phase, the query is flooded in the network and the nodes are organized into an aggregation tree. The base station broadcasting the query is the root of the tree. The query message has a counter that is incremented with each retransmission and counts the hop distance from the root. In this way, each node is assigned to a specific level equal to the node’s hop distance from the root \cite{11}. Also, each sensor chooses one of its neighbours with a smaller hop distance from the root to be its parent in the aggregation tree. The aggregation tree is broadcasted on the network, the total result will arrive at the root. In order to conserve energy, sensor nodes sleep as much as possible during each step where the processor and radio are idle. When a timer expires or an external event occurs, the device wakes up and starts the processing and communication phases. At this point, it receives the messages from its children and then submits the new value(s) to its parent. After that, if no more processing...
2.2 Best-Effort Routing in Sensor Networks:

Recent years have seen significant work on best-effort routing in sensor and other wireless networks. Due to high loss rates and power constraints, a common approach is to use disparity multi-path routing, where more than one copy of a packet is sent to the destination over different paths. For example, directed diffusion uses a flood to discover short paths which sensors would use to send back responses. Various positive and negative reinforcement mechanisms are used to improve path quality. A slightly different approach is used by GRAB, where paths are not explicitly chosen, but the width of the upstream broadcast is controlled.

2.3 Counting Sketches:

Counting sketches were introduced by Flajolet and Martin in for the purpose of quickly estimating the number of distinct items in a database (or stream) in one pass while using only a small amount of space [7]. Since then, there has been much work developing and generalizing counting sketches.

One of the core ideas behind our work is that duplicate insensitive sketches will allow us to leverage the robustness typically associated with multi-path routing. We now present some of the theory behind such sketches and extend it to handle more interesting aggregates. First, we present details of the FM sketches of along with necessary parts of the theory behind them. Then, we generalize these sketches to handle summations, and show that they have almost exactly the same accuracy as FM sketches [7]. We now describe FM sketches for the distinct counting problem.

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

Given a multi-set $M$, the FM sketch of $M$, denoted $S(M)$, is a bitmap of length $k$. The entries of $S(M)$, denoted $S(M)[0, . . . , k-1]$, are initialized to zero and are set to one using a random binary hash function $h$ applied to the elements of $M$. Formally, $S(M)[i] \equiv 1$ 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 $h(x, i) = 1$. This gives a simple serial implementation which is very fast in practice and requires two invocations of $h$ per item on average.

**Theorem 1:** An element $x_i$ can be inserted into an FM sketch in $O(1)$ expected time.

**Algorithm 1 CountInsert $(S,x)$**

1: $i = 0$;
2: while $\text{hash}(x,i) = 0$ do
3: $i = i + 1$;
4: end while
5: $S[i] = 1$;

We now describe some interesting properties of the sketches observed in.

**Property 1** The FM sketch of the union of two multisets is the bit-wise OR of their FM sketches. That is, $S(M_1 \cup M_2)[i] = (S(M_1)[i] \lor S(M_2)[i])$.

**Property 2** $S(M)$ is entirely determined by the distinct items of $M$. Duplication and ordering do not affect $S(M)$.

Property 1 allows each node to compute a sketch of locally held items and send the small sketch for aggregation elsewhere. Since aggregation via union operations is cheap, it may be performed in the network without significant computational burden. Property 2 allows the use of multi-path routing of the sketches for robustness without affecting the accuracy of the estimates. We expand upon these ideas in Section 4. The next lemma provides key insight into the behaviour of FM sketches and will be the basis of efficient implementations of summation sketches later.

3. Our Proposed Method

As mentioned above, building a suitable data set for data mining purposes is a time-consuming task. This task generally requires writing long SQL statements or customizing SQL code if it is automatically generated by some tool. There are two main ingredients in such
SQL code: joins and aggregations; we focus on the second one. The most widely-known aggregation is the sum of a column over groups of rows. Some other aggregations return the average, maximum, minimum or row count over groups of rows. There exist many aggregation functions and operators in SQL. Unfortunately, all these aggregations have limitations to build data sets for data mining purposes. The main reason is that, in general, data sets that are stored in a relational network (or a data warehouse) come from On-Line Transaction Processing (OLTP) systems where network schemas are highly normalized. But data mining, statistical or machine learning algorithms generally require aggregated data in summarized form. Based on current available functions and clauses in SQL, a significant effort is required to compute aggregations when they are desired in a cross tabular (horizontal) form, suitable to be used by a data mining algorithm. Such effort is due to the amount and complexity of SQL code that needs to be written, optimized and tested. There are further practical reasons to return aggregation results in a horizontal (cross-tabular) layout. Standard aggregations are hard to interpret when there are many result rows, especially when grouping attributes have high cardinalities. To perform analysis of exported tables into spreadsheets it may be more convenient to have aggregations on the same group in one row (e.g. to produce graphs or to compare data sets with repetitive information). OLAP tools generate SQL code to transpose results (sometimes called PIVOT). Transposition can be more efficient if there are mechanisms combining aggregation and transposition together. With such limitations in mind, we propose a new class of aggregate functions that aggregate numeric expressions and transpose results to produce a data set with a horizontal layout. Functions belonging to this class are called horizontal aggregations. Horizontal aggregations represent an extended form of traditional SQL aggregations, which return a set of values in a horizontal layout (somewhat similar to a multidimensional vector), instead of a single value per row. This article explains how to evaluate and optimize horizontal aggregations generating standard SQL code.

4. Result and Discussion

We used large wireless sensor data sets and analyzed queries having horizontal aggregation, with different grouping and horizontalization columns. Each experiment was repeated five times and we report the average time in seconds. We cleared cache memory before each method started in order to evaluate query optimization under pessimistic conditions.

This optimization provides a different gain, depending on the method: for SPJ the optimization is best for small n, for PIVOT for large n and for CASE there is rather a less dramatic improvement all across n. it is noteworthy PIVOT is accelerated by our optimization, despite the fact it is handled by the query optimizer. Since this optimization produces significant acceleration for the three methods (at least 2X faster) we will use it by default. Notice that Precomputing FV takes the same time within each method. Therefore, comparisons are fair. We evaluated and optimization specific to the PIVOT operator. This PIVOT optimization is well-known, as we learned from SQL Server DBMS users groups.

Table1-Comparing Evaluation Methods result with time

<table>
<thead>
<tr>
<th>NO OF DATA</th>
<th>SPJ</th>
<th>CASE</th>
<th>PIVOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>62 sec</td>
<td>59 sec</td>
<td>58 sec</td>
</tr>
<tr>
<td>2000</td>
<td>98</td>
<td>97</td>
<td>95</td>
</tr>
<tr>
<td>3000</td>
<td>145</td>
<td>140</td>
<td>138</td>
</tr>
</tbody>
</table>

The above table compares the three query optimization methods. Notice a complement, showing time variability around the mean time μ for times reported in table V we show one show one standard deviation σ and percentage that one σ represents respect to μ. As can be seen, times exhibit small variability, PIVOT exhibits smallest variability, followed by CASE. As we explained before, in time complexity and I/O cost analysis, the two main factors influencing query evaluation time are data set size and grouping columns (dimensions) cardinalities. We consider different combinations of columns to get different values of n and d, respectively. The impact is marginal for all methods at low d.

PIVOT shows a slightly higher impact than the CASE method by the skewed distribution at high d. but overall, both show similar behaviour. SPJ is again the slowest and shows bigger impact at high d. we can find efficiently number of occurrences in a frequent itemsets.
5. Conclusion and Future work

We have presented new methods for approximately computing duplicate-sensitive aggregates across distributed datasets. Our immediate motivation comes from sensor networks, where energy consumption is a primary concern, faults occur frequently, and exact answers are not required or expected. Partitioning large set of data, for the result of horizontal aggregation, in to homogeneous dataset is important task in this system. Association rule of apriori algorithm using SQL is best suited for implementing this operation to find frequent itemset and number of occurrence.

Efficiently evaluating horizontal aggregations using left outer joins presents opportunities for query optimization. Secondary indexes on common grouping columns, besides indexes on primary keys, can accelerate computation. We have shown our proposed horizontal aggregations do not introduce conflicts with vertical aggregations, but we need to develop a more formal model of evaluation. In particular, we want to study the possibility of extending SQL OLAP aggregations with horizontal layout capabilities. Horizontal aggregations produce tables with fewer rows, but with more columns. Thus query optimization techniques used for standard (vertical) aggregations are inappropriate for horizontal aggregations. We plan to develop more complete I/O cost models for cost-based query optimization. We want to study optimization of horizontal aggregations processed in parallel in a shared-nothing DBMS architecture. Cube properties can be generalized to multi-valued aggregation results produced by a horizontal aggregation. We need to understand if horizontal aggregations can be applied to holistic functions (e.g. rank ()). Optimizing a workload of horizontal aggregation queries is another challenging problem.

References


