International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:2, No:8, 2008

A Materialized View Approach to Support Aggregation Operations over Long Periods in Sensor Networks

Minsoo Lee, Julee Choi, and Sookyung Song

Abstract**—**The increasing interest on processing data created by sensor networks has evolved into approaches to implement sensor networks as databases. The aggregation operator, which calculates a value from a large group of data such as computing averages or sums, etc. is an essential function that needs to be provided when implementing such sensor network databases. This work proposes to add the DURING clause into TinySQL to calculate values during a specific long period and suggests a way to implement the aggregation service in sensor networks by applying materialized view and incremental view maintenance techniques that is used in data warehouses. In sensor networks, data values are passed from child nodes to parent nodes and an aggregation value is computed at the root node. As such root nodes need to be memory efficient and low powered, it becomes a problem to recompute aggregate values from all past and current data. Therefore, applying incremental view maintenance techniques can reduce the memory consumption and support fast computation of aggregate values.

Keywords**—**Aggregation, Incremental View Maintenance, Materialized view, Sensor Network.

I. Introduction

SEVERAL recent researches have focused on implementing the sensor network as a database. Such well-known research projects are the COUGAR project [1] which is being carried out at Cornell University and the TinyDB project [2], [3] at UC Berkeley. The architecture proposed by these studies is not the traditional centralized database approach but a network-based approach to compute aggregates in the network whenever possible. That is, the host sends the query to the network and receives the answer from a sensor node. This approach can reduce the number of message transmissions, latency, and power consumption compared to the centralized server-based approach.

Manuscript received July 25, 2006. This work was partially supported by the second stage of Brain Korea 21 program.

Minsoo Lee is with the Dept of Computer Science and Engineering, Ewha Womans University, 11-1 Daehyun-Dong, Seodaemoon-Ku, Seoul, Korea 120-750 (corresponding author phone: +82-2-3277-3401; fax: +82-2-3277-2306; e-mail: mlee@ewha.ac.kr).

Julee Choi is with the Dept of Computer Science and Engineering, Ewha Womans University, 11-1 Daehyun-Dong, Seodaemoon-Ku, Seoul, Korea 120-750 (e-mail: blueri00@ewhain.net).

Sookyung Song is with the Dept of Computer Science and Engineering, Ewha Womans University, 11-1 Daehyun-Dong, Seodaemoon-Ku, Seoul, Korea 120-750 (e-mail: happymint@ewhain.net).

When we form queries we are mostly interested in aggregation values such as the sum, average, and maximum rather than raw sensor readings. Therefore, this paper focuses on aggregation operators and how this operator can be supported in sensor networks. By using incremental view maintenance techniques [4], [5], we can support memory efficient and low powered aggregation operations that can overcome the limitations of sensor nodes.

The organization of the paper is as follows. Section 2 discusses the related work and section 3 gives an overview of the incremental view maintenance for the aggregation operator. Section 4 discusses the implemented system with experimental results and section 5 gives the conclusion with future work.

II. Related Research

The TinyDB and motes which uses the TinyOS operating system has been developed at UC Berkeley. Motes is a small sensor device that also is capable of performing limited computation tasks. The TinyOS provides basic functions to easily develop applications for mote-based ad-hoc networks. Also the Tiny Aggregation (TAG) [6], a power efficient generic aggregation service for ad hoc networks of TinyOS motes, has also been developed.

A few important features of this service are as follows. In TAG, approximate aggregation results are allowed to improve energy efficiency and reduce communication costs. Approximate results are useful for on-line monitoring and can support networking mechanisms for in-network error recovery [7]. And this partial answer enables users to dynamically refine their queries using online-aggregation [8].

The core algorithm in the TAG service is as follows. The parents in the routing tree must produce a single aggregate value that combines the readings of all child nodes in the network during the epoch. If the root node could not provide an aggregate value during the current epoch, many messages must be sent. In order to achieve this, the parents subdivide the epoch and the children are required to deliver their partial state records during a parent-specified time interval. This scheme raises the question of how parents can choose the specified interval in which they will receive the values. If the interval is long enough to receive accurate aggregate values, the power could be wasted. More research needs to be done in this area.

TAG also supports a GROUP BY concept. When parent nodes send a query to their children they also send the predicate which decides the grouping of the children. The children can compute their group and return the group id to their parents. The HAVING clause is also sent to the children so that the answer can be filtered in advance. Additionally, TAG supports several optimization techniques to improve the performance and accuracy. A TAG query has the same structure as an SQL statement except for the EPOCH DURATION clause which means the frequency of obtaining a sensing value. The example below shows a query in TAG which returns the average temperature and id of the sensor at an interval of 30 seconds EPOCH.

```
SELECT AVG (temperature), id
FROM sensors
EPOCH DURATION 30s
```

The aggregation functions in TAG can be categorized based on the characteristics of the sensor network such as duplicate sensitive, exemplary, and monotonic. The following Table I shows such categorizations.

TABLE I
CATEGORIZATION OF AGGREGATION FUNCTIONS IN TAG

	MAX, MIN	COUNT, SUM	AVERAGE
Duplicate Sensitive	No	Yes	Yes
Exemplary(E),Summary(S)	E	S	S
Monotonic	Yes	Yes	No

Materialized views [4], [9] have been extensively researched in the data warehousing area. They are used to improve the database performance by working in a similar way as an index. Materialized views actually store records that result from the computation of the materialized view definition. Therefore, the query is run against the records of the materialized view rather than the base tables, and this dramatically increases the performance of the query processing because the records in the view need not be recomputed [10]. One drawback of materialized views is that they need to be maintained and refreshed. Changes to the base relations cause the recomputation of the view and recomputing the view from scratch can waste time and power. Therefore, incremental view maintenance is the desired way for maintaining materialized views by computing only the changes that should be applied to the view. Most of the view maintenance mechanisms use mathematical expressions to define a view and compute the changes to the view.

III. INCREMENTAL VIEW MAINTENANCE FOR AGGREGATION OPERATORS IN SENSOR NETWORKS

A. Current Technology to Calculate Aggregations in Sensor Networks

The steps to compute aggregation values in the current sensor network setting are as follows. First, the sensor network organizes the routing tree to send the user's query to the sensor nodes. Then, the host sends the query with the predicate to group the nodes. Each node sends the query to its child node and gets the result from the child node and returns the result and its group value to its parent node.

The results of an aggregation query would have information such as <group id, aggregation value>. This aggregation value is calculated from sensing data in the same epoch. Aggregation functions are composed of a function ***f*** for merging, a function ***I*** for initialization and a function ***e*** for evaluation.

Fig. 1 shows the current method of performing calculation of the aggregation operator AVG in a sensor network. First each sensor (node 1, 2, 3) senses and initializes the value. Then, these values are merged along the routing tree. At the end, the final node (node 4) calculates the average value.

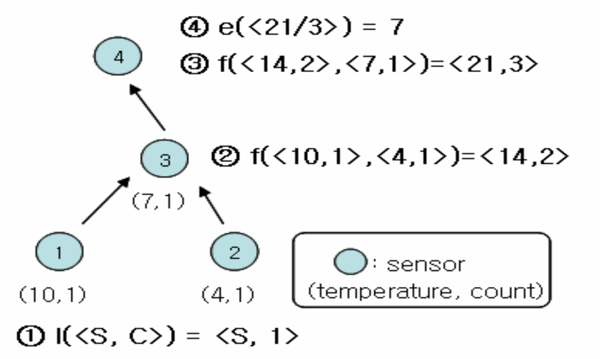


Fig. 1 Current technology to calculate aggregation functions

B. Adding the DURING Clause

In the current sensor network database system it is not possible to obtain the aggregation value over a long period of time because the sensors do not have the power or memory to accumulate the sensed data for a long period of time. In order for us to support this kind of aggregation we first propose to add a DURING clause to the TinySQL syntax. The following query is an example using the DURING clause and Table II shows the syntax and various usages for the DURING clause.

```
SELECT AVG (temperature), id
FROM sensors
EPOCH DURATION 30s
DURING 10hr
```

TABLE II
THE DURING CLAUSE SYNTAX

DURING Syntax	Example	Description
start – end	6:00 - 16:00	From 6 to 16
start[interval]	6:00 [10hr]	From 6, during 10
[interval]*	[10hr]*	Every 10
interval	10hr	During 10
Interval epoch	100 epoch	During 10 epoch times

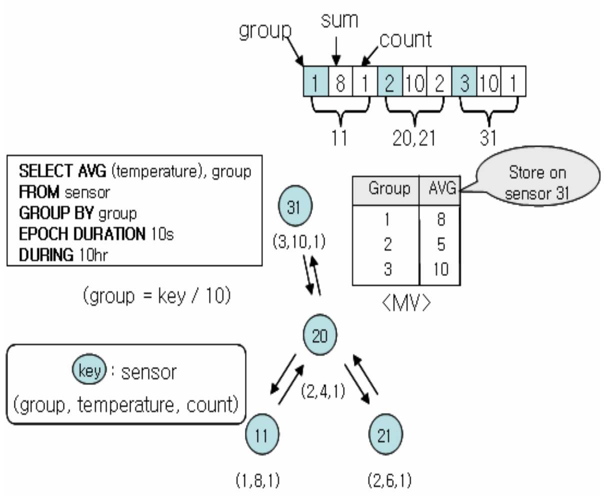
C. Query Processing and Incremental View Maintenance

To calculate the aggregation value based on the DURING clause, we propose to use materialized view techniques and incremental view maintenance.

Assuming that sensors in the same area have the same group id, and we want the average temperature values of each group, we can define a materialized view using the extended TinySQL and create it at the base station node for efficient computation.

```
CREATE MATERIALIZED VIEW
                    V (AVG (temperature), group) AS
   (SELECT AVG (temperature), group
    FROM sensors
    GROUP BY group
    DURING 10hr)
```

As shown in Fig. 2, the query computes the average temperature values in groups during a period of 10 hours. When the parent nodes send the query, they also send the predicate to decide the group they belong to. Child nodes receive this query and predicate, and choose their own group and return the aggregate data records (group id, sensing data, count). When a node receives an aggregate from a child, it calculates the sum of the temperature and the sum of counts for the related group and returns the result. Finally, the base station sensor (node 31) calculates the average and stores the data into the materialized view. According to the predicate (group = key / 10), the group id of node 31 is 3, the group id of node 21 is 2, and the group id of node 11 is 1.



```
SELECT AVG (temperature), group
FROM sensor
GROUP BY group
EPOCH DURATION 10s
DURING 10hr
```

Fig. 2 Supporting long-duration queries with current technology

Using this method, during all epochs, we need to store all the sensing data as shown in Fig. 3. Tn means the n'th epoch from T1 and the sensor node 31 stores all data to calculate the average value during the n epochs. At Tn in order to recalculate the materialized view, the sensing data organized in groups at each epoch (from T1 to Tn) are needed. Therefore if the time period is much longer, the system and sensor nodes will experience more overhead. Thus, considering the limitations of the sensor nodes, this approach is impossible.

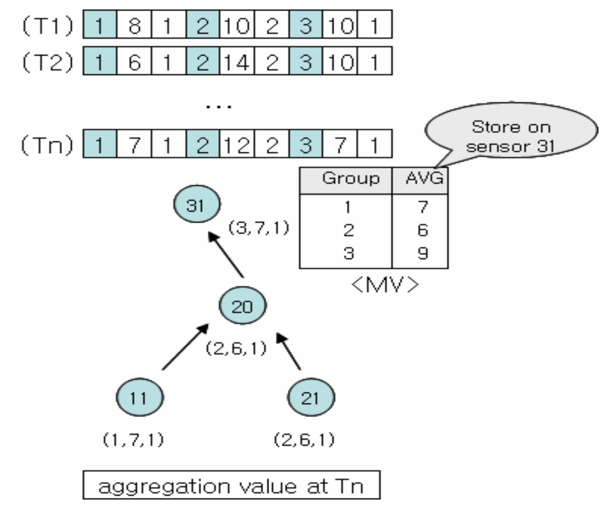


Fig. 3 Overhead without Incremental View Maintenance

Our work suggests that the incremental view maintenance approach be taken to efficiently update the materialized view. The following statement shows the formula for incremental view maintenance. Using this formula, the changes (Δ) on the base relations can be collectively calculated and applied to the view [6].

$$
\begin{aligned}
\Delta V = {} & (\Delta S1 \infty S2 \infty S3 \ \ldots \infty Sn) \cup \\
& (S1' \infty \Delta S2 \infty S3 \ldots \infty Sn) \cup \\
& \ldots \\
& (S1' \infty S2' \infty S3' \ldots \infty \Delta Sn) \\
V' = {} & V \cup \Delta V
\end{aligned}
$$

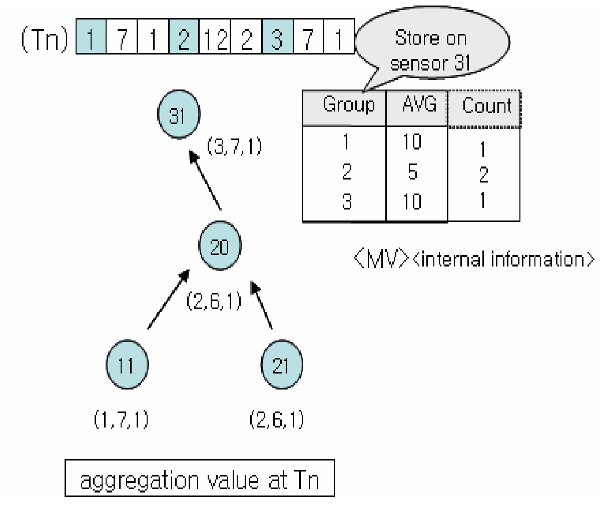


Fig. 4 Query processing with Incremental View Maintenance

Fig. 4 shows the query processing for the DURING clause using incremental view maintenance in comparison with Fig. 3.

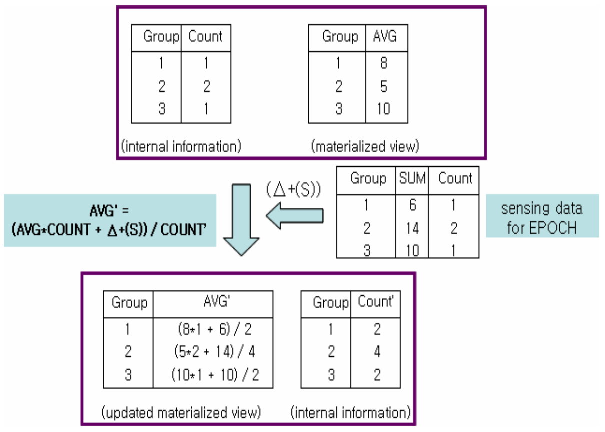
Table III shows the formula used for several aggregation operators to recalculate the materialized view using incremental view maintenance. Δ+(S) is the insertion value to apply to the view, or in other words the sensing result of the query at each epoch and MAX' means the recalculated MAX value.

TABLE III
INCREMENTAL VIEW MAINTENANCE FORMULAS

MAX	if Δ+(S) > MAX MAX' = Δ+ (S)
MIN	if Δ+(S) < MIN MIN' = Δ+(S)
SUM	SUM' = SUM + Δ+(S)
COUNT	COUNT' = COUNT + Δ+(S)
AVG	AVG' = (AVG*COUNT + Δ+(SUM)) / (COUNT + Δ+(COUNT))

Fig. 5 shows the recalculated views using Table III formulas. In the case of group 1, the AVG value at the former epoch is 8 and the COUNT value is 1. And at the current epoch, Δ + (Sum) value is 6 and Δ + (Count) value is 1. Using the incremental view maintenance, the average value of the group 1 is 7.

$$\boxed{AVG' = (AVG * COUNT + \Delta + (SUM)) / (COUNT + \Delta + (COUNT) = (8*1 + 6) / 2 = 7}$$



Group	Count
1	1
2	2
3	1

(internal information)

Group	AVG
1	8
2	5
3	10

(materialized view)

Group	SUM	Count
1	6	1
2	14	2
3	10	1

Group	AVG'
1	(8*1 + 6) / 2
2	(5*2 + 14) / 4
3	(10*1 + 10) / 2

(updated materialized view)

Group	Count'
1	2
2	4
3	2

(internal information)

Fig. 5 Materialized view recomputation using incremental view maintenance

However, using materialized views for efficiently using memory and reducing the updating overhead for a long duration aggregation query may not be effective in some situations. For example, if the query is to calculate the average temperature during 3 minutes, using materialized views is not effective because sensor node may have enough memory and do not need to additionally take the updating overhead of materialized views.

IV. Implementation and Experimental Results

In this section, we evaluate the performance of our algorithm using TinyOS and TinyDB.

A. System Environment and Implementation

Ustar2400 is a hardware platform for sensor networks developed by the HUINS Company. Fig. 6 shows the structure of Ustar2400. In the Ustar2400 board, sensor nodes return the data to the base station node. Using the ISP board, the base station node can be connected to the host computer and transfer data. We have implemented our incremental view maintenance algorithm in the TinyDB at the host computer. The latest version of TinyOS is tinyos-1.x and has TinyOS, TinyDB and cygwin together in an integrated development environment. TinyDB is implemented in JAVA and it supports a GUI or command window to send queries to the host server.

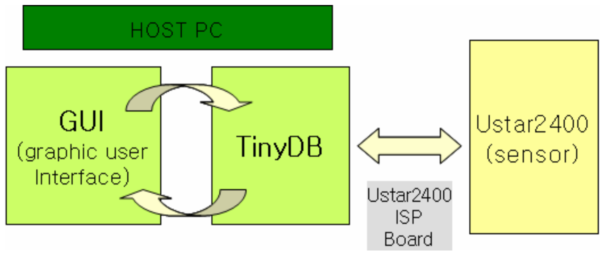
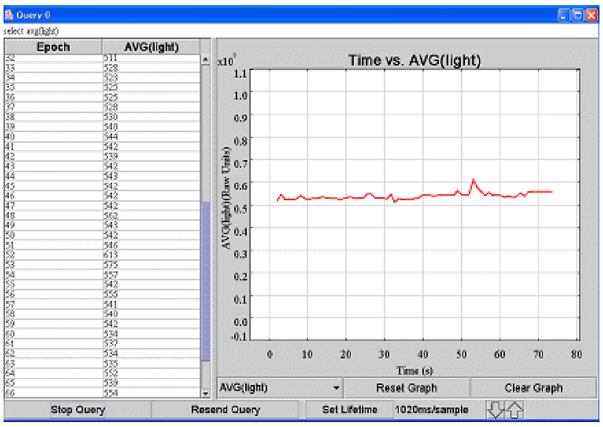


Fig. 6 Structure of UStar2400 system

The TinyDB JAVA API main classes are TinyDBNetwork, SensorQueryer, TinyDBQuery, QueryResult, AggOp. The main classes of TinyDB GUI API are TinyDBMain, CmdFrame, MainFrame. We have changed and extended the QueryResult and AggOP classes. By changing these classes we can define a materialized view and recompute the view using incremental view maintenance techniques as discussed earlier. Additionally, the AverageReader, IntReader, AggregateResultReader classes were modified by adding new functions.

B. Experimental Results

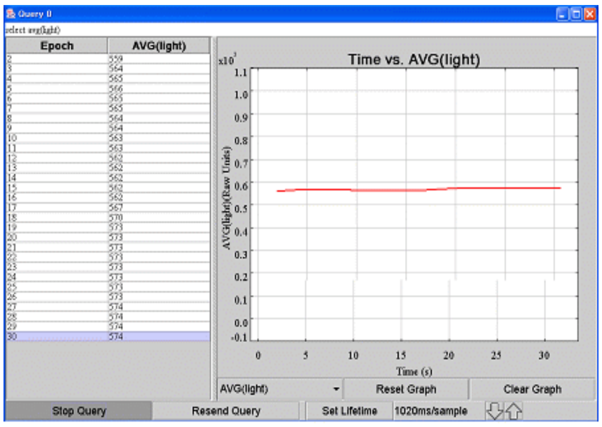
Fig. 7 shows the graph showing the calculation of averages using the previous TinyDB. The number of sensor nodes in each epoch is not always consistent. This means that in each epoch the average value creates some fluctuations. Fig. 8 shows the graph for calculating the average during 10 hours using the TinyDB with incremental view maintenance. As time goes by, the aggregation value is consistent and does not show any big fluctuation because of its cumulative nature realized by the materialized view.



select avg(light)

Epoch	AVG(light)
32	511
33	528
34	523
35	525
36	525
37	528
38	530
39	540
40	544
41	542
42	539
43	542
44	543
45	542
46	542
47	542
48	562
49	543
50	542
51	546
52	613
53	575
54	557
55	542
56	555
57	541
58	540
59	542
60	534
61	537
62	534
63	535
64	552
65	539
66	554

Fig. 7 Aggregation result using previous TinyDB



select avg(light)

Epoch	AVG(light)
2	559
3	564
4	565
5	566
6	565
7	565
8	564
9	564
10	563
11	563
12	562
13	562
14	562
15	562
16	562
17	567
18	570
19	573
20	573
21	573
22	573
23	573
24	573
25	573
26	573
27	574
28	574
29	574
30	574

Fig. 8 Aggregation result using incremental view maintenance

V. Conclusion

When sensor networks are implemented as sensor databases, most of the queries focus on aggregation operators because the user's interest is on the summarized values rather than the raw data. We have looked into ways to intelligently and efficiently execute aggregation queries in the sensor network as a core service.

We have especially studied the latest techniques, incremental view maintenance, which is used in the data warehousing area and applied this to aggregation grouping in the sensor network. This can improve the energy efficiency and reduce the sensor memory limitation.

For future work, we need to perform further research on experiment with specific cases for finding the boundaries on where the incremental view maintenance technique may incur more overhead than benefit for specific cases. Extensive performance evaluation of the incremental view maintenance for aggregation grouping will be carried out and more optimization techniques for aggregation service in sensor networks will be studied.

References

[1] Y. Yao and J. Gehrke, "The Cougar approach to in-network query processing in sensor networks," in Proc. *ACM SIGMOD*, 2002

[2] S.Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "The design of an acquisational query processor for sensor networks," *ACM SIGMOD,* 2003

[3] TinyDB, http://berkeley.intel-research.net/tinydb/

[4] Ashish Gupta and Inderpal Singh Mumick, "Maintenance of Materialized View: Problems, Techniques, and Applications," *IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Data Warehousing,* Vol. 18, No. 2, 1995

[5] Ki Yong Lee, Jin Hyun Son and Myoung Ho Kim, "Efficient Incremental View Maintenance in Data Warehouses," *ACM CIKM,* 2001

[6] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong, "TAG: a Tiny Aggregation Service for Ad-Hoc Sensor Networks," *ACM SIGOPSI,* Vol. 36, pp. 131-146, 2002

[7] Ramesh Govindan, Joseph M. Hellerstein, Wei Hong, Samuel Madden, Michael Franklin, and Scott Shenker, "The Sensor Network as a Database," Technical Rep ort 0-771, Compm2E Science Department

[8] Joseph M. Hellerstein, Peter J. Haas, and Helen J. Wang., "Online Aggregation," *ACM SIGMOD International Conference on Management Data (ICDM),* 1997

[9] Y.Zhuge, H. Garcia-Molina, J. Hammer, and J. Widom, "View Maintenance in a warehousing environment," *ACM SIGMOD,* 1994

[10] Ashish Gupta, Inderpal Singh Mumick, V.S. Subrahmanian. , "Maintaining Views Incrementally," *ACM SIGMOD Record,* Vol. 22, 1993, p157-166

[11] Amit Manjhi, Suman Nath, Phillip B. Gibbons, "Tributaries and Deltas: Efficient and Robust Aggregation in Sensor Network Streams," *ACM SIGMOD,* 2005

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