Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2013, Article ID 512368, 10 pages
http://dx.doi.org/10.1155/2013/512368

Research Article

Spatial TinyDB: A Spatial Sensor Database System for the USN Environment

Dong-Oh Kim,[1] Lei Liu,[2] In-Su Shin,[3] Jeong-Joon Kim,[3] and Ki-Joon Han[3]

[1] *Cloud Computing Research Department, Electronics and Telecommunications Research Institute, Daejeon 305-700, Republic of Korea*
[2] *Domestic Financial Department, China Banking Regulatory Commission Henan Office, 6 Cuizhu Street, High & New Technology Industries Development Zone, Zhengzhou 450000, China*
[3] *Division of Computer Science & Engineering, Konkuk University, Seoul 143-701, Republic of Korea*

Correspondence should be addressed to Jeong-Joon Kim; jjkim9@db.konkuk.ac.kr

Received 13 January 2013; Accepted 19 July 2013

Academic Editor: Lei Shu

For the Ubiquitous Sensor Network (USN) environment, which generally uses spatial as well as aspatial sensor data, a sensor database system to manage these data is essential. For this reason, sensor database systems such as TinyDB and Cougar are being developed by researchers. However, as most of these systems do not support spatial data types and spatial operators for managing spatial sensor data, they are not suitable for the USN environment. Therefore, in this paper, we design and implement Spatial TinyDB which is a spatial sensor database system that extends TinyDB to support spatial data types and spatial operators for the efficient management of spatial sensor data. In particular, Spatial TinyDB provides memory management and filtering functions to reduce system overload caused by sensor data streams. Finally, we prove that Spatial TinyDB is superior by comparing its actual performance, in terms of execution time, accuracy, and memory usage, with that of TinyDB.

1. Introduction

With the development of sensor technologies for sensing various types of data (such as temperature, humidity, and pressure) and with advances in wireless communication technologies (resulting in technologies such as CDMA, WiFi, and WiBro), there is increasing interest in, and research on, the application of technologies related to ubiquitous sensor networks (USNs) in areas such as ecosystem monitoring, home automation, and car theft detection [1].

A USN is a communication network in which various types of sensor nodes interconnected by means of wireless communication schemes manage sensed data [2, 3]. A sensor node consists of sensing, processing, storage, and communication modules. However, it has limited hardware and software capacities. That is, it has limitations with regard to its capacity to process sensed data, the space available to store sensed data, the distance of data transmission, and the amount of electric power available. In particular, the sensor node consumes more power for data transmission than for data processing.

A geosensor can obtain its location, either directly or indirectly, via RFID readers, GPSs, CCTVs, and so forth, and can generate various forms of related stream data [4]. The use of geosensors is increasing, particularly in the USN environment, as they are utilized in the provision of diverse services related to u-GIS, u-LBS, u-Logistics, u-Transportation, u-Medicine, u-Disaster Prevention, and so forth. In this sense, geosensors are leading the ubiquitous age using both aspatial and spatial data simultaneously. Accordingly, there is active research into the efficient management of spatial sensor data collected by geosensors in the USN environment [5].

Recently, various sensor database systems, including TinyDB [6] and Cougar [7], have been developed for the efficient management of sensor data in the USN environment due to the fact that existing sensor database systems do not support spatial data types and spatial operators. However, these recently developed systems cannot efficiently manage spatial sensor data from geosensors. In addition, SE TinyDB [8]—a spatial extension of TinyDB—provides its own spatial operators but cannot support the international standards recommended by the Open Geospatial Consortium (OGC).

In this paper, we design and implement Spatial TinyDB which is a spatial sensor database system that provides various spatial data types and spatial operators for the efficient management of spatial sensor data in the USN environment. In particular, Spatial TinyDB extends TinyDB—an existing sensor database system—to facilitate the efficient management of spatial sensor data. It also conforms to the Simple Feature Specification for SQL [9], a standard proposed by the OGC, in extending spatial data types and spatial operators for interoperability.

This paper is organized as follows. Following the introduction given in Section 1, Section 2 analyzes related studies on the TinyDB and SE TinyDB sensor database systems. Section 3 describes the overall structure of Spatial TinyDB and outlines each of the managers used in Spatial TinyDB. Section 4 verifies the superiority of Spatial TinyDB by means of performance evaluations. Finally, we give concluding remarks in Section 5.

2. Related Works

In this section we analyze TinyDB (used in the implementation of Spatial TinyDB proposed in this paper) and look at SE TinyDB—a spatial extension of TinyDB.

2.1. TinyDB. TinyDB [6, 10] is a query processing system used in TinyOS to extract information from wireless sensor networks. TinyOS [11]—an open source system developed at the University of California Berkeley using nesC language—is the most widely used representative sensor operating system. It modularizes the system into component units and each component is connected to other components through function calls known as interfaces. Thus, application developers can develop applications using components as libraries and connect the components through interfaces.

The characteristics of TinyDB are as follows. First, it provides a metadata catalog for describing the types of sensors in a sensor network. Second, it supports a query language that easily describes the data desired by users. Third, it can form a network by itself for query processing. Fourth, it can process multiple queries for multiple sensor nodes simultaneously. Fifth, a TinyDB sensor network can be extended by simply downloading a standard TinyDB code onto a new node and resetting the node.

Query processing in TinyDB is as follows. First, an input query from the server PC is transmitted to the network as an optimized query. The node that receives the transmitted query then acquires data from its neighboring nodes based on a routing tree and executes aggregate operations within the network. Finally, the result is returned to the server PC.

In order to enable users to extract the data desired without doing any programming, TinyDB provides a simple interface that is similar to SQL. Table 1 shows the query statement format supporting the SQL-like interface and a typical query example that can be used in TinyDB.

As shown in Table 1, clauses SELECT, WHERE, GROUP BY, and HAVING in the query statement format are similar to those of standard SQL. TRIGGER ACTION executes triggered actions such as SOUND and LED when the conditions of the WHERE statement are satisfied. EPOCH DURATION is used to set the time interval between epochs, in millisecond (ms). The example query in Table 1 gets the mean temperature of sensor nodes whose temperatures exceed 100°C. Table 2 lists the data types and operators supported by TinyDB.

Table 1: Query statement format and query example.

Query statement format	SELECT select-list ⟨FROM sensors⟩ WHERE where-clause ⟨GROUP BY gb-list⟩⟨HAVING having-list⟩ ⟨TRIGGER ACTION command-name ⟨(param)⟩⟩ ⟨EPOCH DURATION⟩
Query example	SELECT AVG (temp) FROM sensors WHERE temp > 100

Table 2: Data types and operators.

Data types	Operators
int8, int16, int32, uint8, uint16, uint32, string	SUM, AVERAGE, MIN, MAX, COUNT

Table 3: Spatial operators in SE TinyDB.

Operators	Explanation
DISTANCE	DISTANCE [ID, X, Y]
INBOX	INBOX [$X_{\min}, Y_{\min}, X_{\max}, Y_{\max}$]
BEYONDBOUNDARY	BEYONDBOUNDARY [$X_{\min}, Y_{\min}, X_{\max}, Y_{\max}$, CMD (par)]

As can be deduced from Table 2, TinyDB does not support spatial data types and spatial operators. As a result, it has difficulty in managing spatial data efficiently.

2.2. SE TinyDB. SE TinyDB [8] was designed and developed to process both spatial and aspatial queries in sensor nodes. SE TinyDB extends traditional TinyDB by adding the spatial operators DISTANCE, INBOX, and BEYONDBOUNDARY. Table 3 lists the spatial operators used to extend TinyDB to develop SE TinyDB.

In Table 3, the DISTANCE operator returns the coordinate distance between sensor nodes, while the INBOX operator returns the location of the sensor nodes within a specified rectangle. The BEYONDBOUNDARY operator returns the location of sensor nodes outside a specified rectangle.

Table 4 gives examples of spatial queries that use the spatial operators in SE TinyDB.

In Table 4, Example 1 returns the ID (nodeid), temperature (temp), and location coordinates (lat, lon) of those sensor nodes less than 200 units away from a specified point (Point (500, 500)). Example 2 returns the ID (nodeid), temperature (temp), and location coordinates (lat, lon) of those sensor nodes located within a specified rectangle (0, 0, 500, 500). Finally, Example 3 returns the ID (nodeid), temperature

Table 4: Examples of spatial query in SE TinyDB.

Example 1	SELECT nodeid, temp, lat, lon FROM sensors WHERE DISTANCE [nodeid, 500, 500] < 200
Example 2	SELECT nodeid, temp, lat, lon FROM sensors WHERE INBOX [0, 0, 500, 500]
Example 3	SELECT nodeid, temp, lat, lon FROM sensors WHERE BEYONDBOUNDARY [0, 0, 500, 500]

(temp), and location coordinates (lat, lon) of those sensor nodes located outside a specified rectangle (0, 0, 500, 500).

The three spatial operators shown in Table 3 were incorporated into SE TinyDB to facilitate the processing of queries on a specific area or moving trajectory. However, SE TinyDB does not support various spatial data types and spatial operators related to the Simple Feature Specification for SQL, the standard recommended by the OGC.

3. Spatial TinyDB

In this section, we explain the overall structure of Spatial TinyDB and look at the various managers comprising it.

3.1. System Structure. The proposed Spatial TinyDB extends TinyDB to facilitate the efficient management of spatial sensor data in USN environments. Figure 1 depicts the overall structure of Spatial TinyDB.

As depicted in Figure 1, Spatial TinyDB consists of a spatial data manager, a spatial query processing manager, a spatial data stream manager, a spatial interface manager, and a spatial data communication manager. In Figure 1, the light-blue boxes signify the extended TinyDB components, while the pink boxes signify newly added components.

The spatial data manager manages spatial data types in conformance with the OGC standards and converts spatial attribute information into spatial schema. The spatial query processing manager provides spatial relation operators and spatial analysis operators in line with the OGC standards. In addition, it provides spatial trajectory operators for processing the moving trajectories of sensor nodes. The spatial data stream manager provides memory sharing functions among spatial queries in processing spatial queries and filtering functions for reducing input load. The spatial interface manager receives spatial queries, parses them, and then displays the final results. Finally, the spatial data communication manager manages communication and sessions between the server PC and sensor nodes in a wireless sensor network and between sensor nodes.

3.2. Spatial Data Manager. In this subsection, we look at the spatial data type management module and the spatial schema conversion module forming the spatial data manager.

Table 5: Spatial data types and examples.

Spatial data types	Examples
Point	POINT (10 10)
LineString	LINESTRING (10 10, 20 20, 30 40)
Polygon	POLYGON (10 10, 10 20, 20 20, 20 15, 10 10)
PolyhedralSurface	POLYHEDRALSURFACE ((10 10, 10 20, 20 20, 20 10, 10 10), (20 10, 40 20, 20 20, 20 10))
MultiPoint	MULTIPOINT (10 10, 20 20)
MultiLineString	MULTILINESTRING ((10 10, 20 20), (15 15, 30 15))
MultiPolygon	MULTIPOLYGON ((10 10, 10 20, 20 20, 20 15, 10 10), (60 60, 70 70, 80 60, 60 60))

3.2.1. Spatial Data Type Management Module. The spatial data type management module provides the spatial data types recommended in the "Simple Features Specification for SQL" Standard Specifications [9] of the OGC in order to support spatial queries. Table 5 shows spatial data types and examples supported in the spatial data type management module.

As shown in Table 5, the spatial data type management module supports seven spatial data types: Point, LineString, Polygon, PolyhedralSurface, MultiPoint, MultiLineString, and MultiPolygon.

3.2.2. Spatial Schema Conversion Module. The spatial schema conversion module converts spatial attribute information sent by the spatial data communication manager into spatial schema according to spatial schema mapping rules. Figure 2 illustrates the spatial schema conversion process.

As depicted in Figure 2, the spatial schema conversion module converts the spatial attribute information Location, Lines, and Boundary into Point, LineString, and Polygon, respectively, according to spatial schema mapping rules.

3.3. Spatial Query Processing Manager. In this subsection, we look at the spatial relation operator module, the spatial analysis operator module, and the spatial trajectory operator module forming the spatial query processing manager.

3.3.1. Spatial Relation Operator Module. The spatial relation operator module provides the spatial relation operators recommended in "Simple Features Specification for SQL" Standard Specifications [9] of the OGC, in order to support spatial query processing in the server PC and sensor nodes. Table 6 shows the spatial relation operators provided in the spatial relation operator module.

As can be seen in Table 6, the spatial relation operator module supports eight spatial relation operators: Equals, Disjoint, Touches, Within, Overlaps, Crosses, Intersects, and Contains. A spatial relation operator receives two spatial objects, Geometry A and Geometry B, as the input and returns True or False as the output.

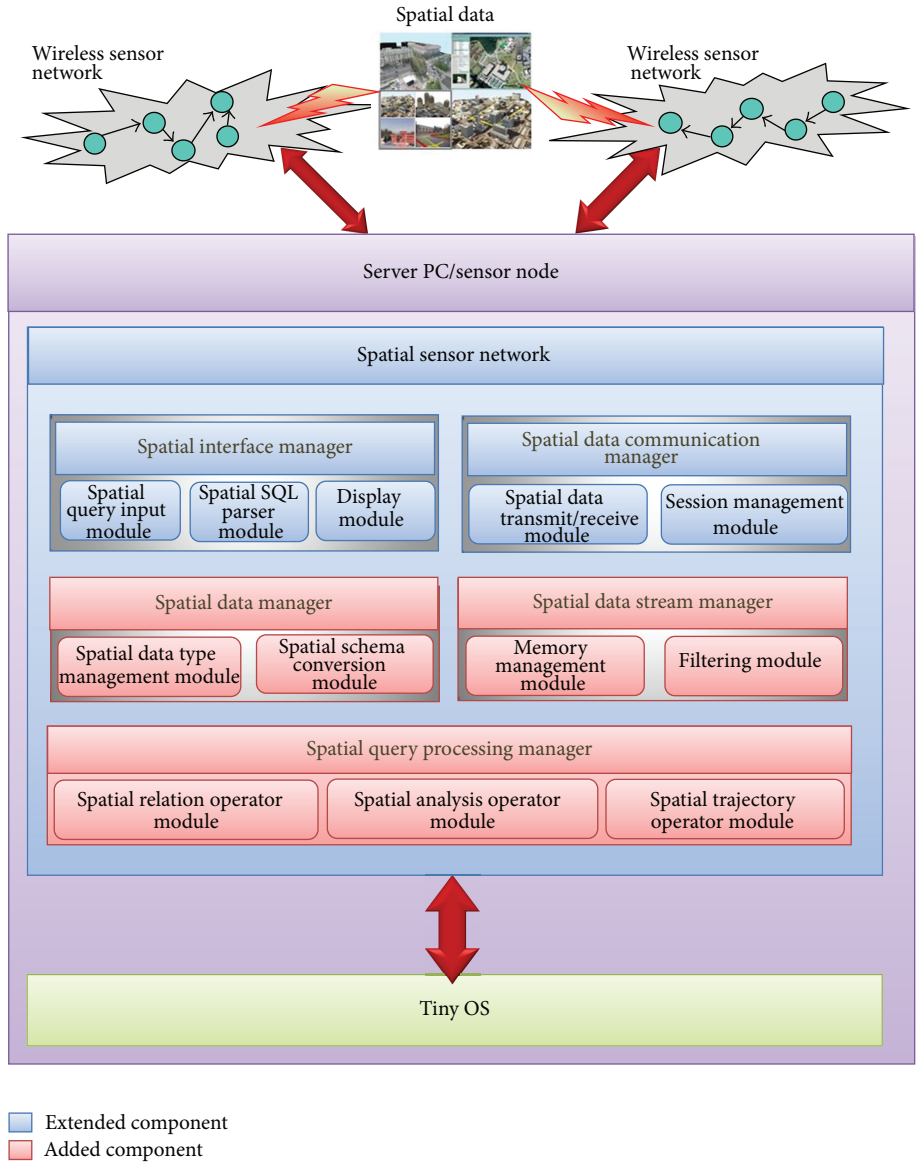

Figure 1: Structure of Spatial TinyDB.

3.3.2. Spatial Analysis Operator Module. The spatial analysis operator module provides the spatial analysis operators recommended in the "Simple Features Specification for SQL" Standard Specifications [9] of the OGC, in order to process spatial queries in the server PC and sensor nodes. Table 7 lists the spatial analysis operators provided in the spatial analysis operator module.

As shown in Table 7, the spatial analysis operator module supports six spatial analysis operators: Distance, Intersection, Difference, Union, Buffer, and ConvexHull. Each spatial analysis operator receives two spatial objects, Geometry A and Geometry B, as the input and returns a new spatial object as the output.

3.3.3. Spatial Trajectory Operator Module. The spatial trajectory operator module provides spatial trajectory operators for processing the moving trajectories of sensor nodes [12–14]. Table 8 lists the spatial trajectory operators provided in the spatial trajectory operator module.

As listed in Table 8, the spatial trajectory operator module supports five spatial trajectory operators: Enter, Insides, Leaves, Meets, and Passes. A spatial trajectory operator receives two spatial objects, Geometry A and Geometry B, as the input and returns True or False as the output.

3.4. Spatial Data Stream Manager. In this subsection, we look at the memory management module and the filtering module comprising the spatial data stream manager.

3.4.1. Memory Management Module. The memory management module provides data sharing functions among various spatial queries executed in Spatial TinyDB in order to reduce the system load caused by spatial data streams [5, 15].

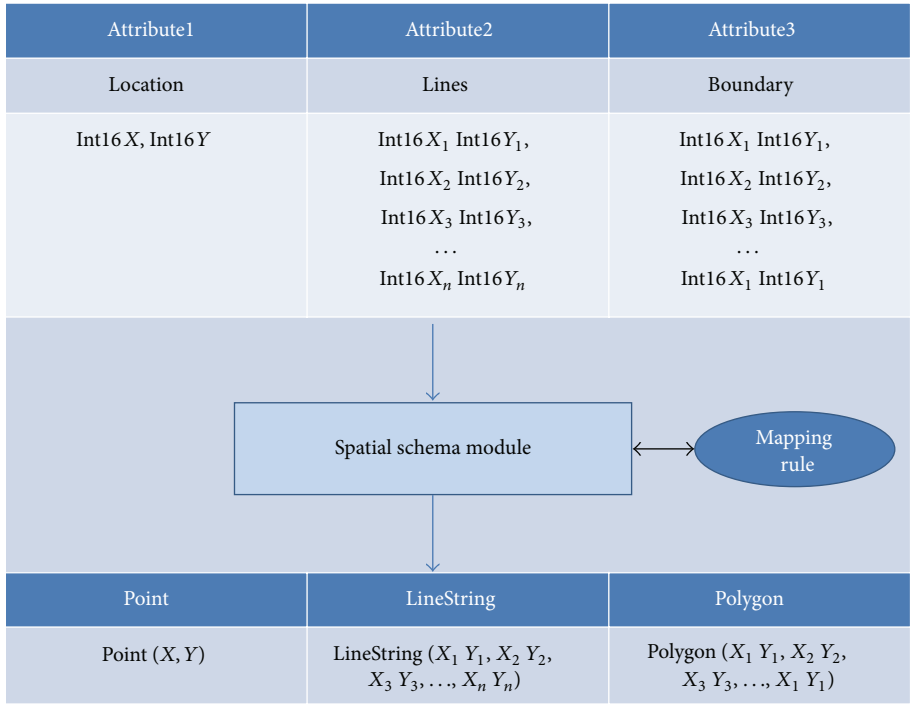

Figure 2: Spatial schema conversion process.

Table 6: Spatial relation operators.

Spatial relation operators	Explanation
Equals (Geometry A, Geometry B)	Return whether or not Object A is equal to Object B
Disjoint (Geometry A, Geometry B)	Return whether or not Object A is apart from Object B
Touches (Geometry A, Geometry B)	Return whether or not the boundary of Object A meets the boundary of Object B
Within (Geometry A, Geometry B)	Return whether or not Object A is included in Object B
Overlaps (Geometry A, Geometry B)	Return whether or not Object A and Object B overlap each other
Crosses (Geometry A, Geometry B)	Return whether or not Object A crosses Object B
Intersects (Geometry A, Geometry B)	Return whether or not Object A intersects Object B
Contains (Geometry A, Geometry B)	Return whether or not Object A contains Object B

Table 7: Spatial analysis operators.

Spatial analysis operators	Explanation
Distance (Geometry A, Geometry B)	Return the distance between Object A and Object B
Intersection (Geometry A, Geometry B)	Return the intersection of Object A and Object B
Difference (Geometry A, Geometry B)	Return the difference between Object A and Object B
Union (Geometry A, Geometry B)	Return the union of Object A and Object B
Buffer (Geometry A, Double L)	Return a spatial object whose boundary is larger by L from the boundary of Object A
ConvexHull (Geometry A)	Return the smallest convex polygon that can contain Object A

When Spatial TinyDB executes two or more spatial queries simultaneously, the spatial queries share one memory area instead of having their own respective memory areas. In the shared memory, a reference counter is set for each data tuple. If a spatial query processes a data tuple, the reference counter corresponding to the data tuple is reduced by 1, and if the reference counter becomes 0, the corresponding data tuple is deleted from the memory.

For example, if Spatial Query 1 saves a specific tuple in the depository and Spatial Query 2 and Spatial Query 3 share the tuple, the tuple's reference counter becomes 3. If Spatial Query 1 processes the tuple, the tuple is not deleted from the depository, but its reference counter is reduced by 1. If both Spatial Query 2 and Spatial Query 3 process the tuple, the reference counter becomes 0 and the corresponding tuple is deleted.

3.4.2. Filtering Module. The filtering module provides filtering functions that solve the overload problem by reducing the volume of the input data stream, while minimizing loss of accuracy [16]. It carries out filtering of the input data stream using filtering conditions such as difference in the distance of location coordinates, time range at specific times, and IDs of sensor nodes. It then delivers only the filtered data stream to the spatial query processing manager. Figure 3

Table 8: Spatial trajectory operators.

Spatial trajectory operators	Explanation
Enter (Geometry A, Geometry B)	Return whether or not Object A enters Object B from outside
Insides (Geometry A, Geometry B)	Return whether or not Object A stays inside Object B
Leaves (Geometry A, Geometry B)	Return whether or not Object A goes out of Object B from inside
Meets (Geometry A, Geometry B)	Return whether or not Object A only touches the boundary of Object B
Passes (Geometry A, Geometry B)	Return whether or not Object A enters Object B from outside and then goes out of Object B from inside

Table 9: Spatial query examples.

Query types	Examples
Spatial relation operators	SELECT nodeid, temp, loc FROM sensors WHERE Contains (polygon (0 0, 40 0, 40 40, 0 40, 0 0), loc)
Spatial analysis operators	SELECT nodeid, temp, loc FROM sensors WHERE Contains (Intersection (Polygon (0 0, 0 40, 40 40, 40 0, 0 0), Polygon (30 30, 30 70, 70 70, 70 30, 30 30), loc)
Spatial trajectory operators	SELECT nodeid, temp, loc FROM sensors WHERE Passes (loc, Polygon (0 0, 0 40, 40 40, 40 0, 0 0))

gives an example of spatial data filtering in accordance with a filtering condition that consists of IDs of sensor nodes and the difference in the distance of location coordinates.

As illustrated in Figure 3, if "ID is from 1 to 5 and the difference in the distance of location coordinates is less than 100" is the filtering condition, the sensor node filters out the input data stream with IDs from 1 to 5 and with the difference in distance less than 100 between the previous location and the current location of the same object, and it delivers only the filtered data stream to the spatial query processing manager.

3.5. Spatial Interface Manager. In this subsection, we explain the spatial query input module, the spatial SQL parser module, and the display module forming the spatial interface manager.

3.5.1. Spatial Query Input Module. The spatial query input module receives spatial queries from the user and delivers them to the spatial SQL parser module. In addition, it chooses between text and graphic modes in displaying the results of a spatial query and receives the parameters (execution cycle, query ID, query condition, etc.) of a spatial query.

In Spatial TinyDB, the format of the spatial query statements is the same as that used in TinyDB. However, it can also use spatial relation operators, spatial analysis operators, and spatial trajectory operators, in addition to spatial data types, when building a spatial query. Table 9 shows spatial query examples that are impossible in TinyDB but possible in Spatial TinyDB.

In Spatial TinyDB, various spatial queries can be built by using spatial data types, spatial relation operators, spatial analysis operators, and spatial trajectory operators, as depicted in Table 9.

3.5.2. Spatial SQL Parser Module. The spatial SQL parser module parses a spatial SQL statement received from the spatial query input module (by executing lexical and syntactic analyses) and tests the validity of the spatial SQL statement based on parsed information. It also does error processing when a spatial SQL statement is incorrect. In addition, for a spatial query for which spatial SQL parsing has been completed normally, the module creates spatial attribute and spatial operator information and delivers them to the spatial data communication manager.

3.5.3. Display Module. The display module displays the status of query execution, final query results, and so forth, on the screen of the server PC. In particular, the module displays final query results received from the spatial query processing manager in two modes—text and graphic. In addition, it can display spatial SQL query statements received from the spatial SQL parser module and error messages for errors occurring while a spatial query is being executed. Further, the display module allows the user to reset the query execution cycle during the execution of a spatial query and to control operations such as pausing and restarting a spatial query.

3.6. Spatial Data Communication Manager. In this subsection, we look at the session management module and the spatial data transmit/receive module forming the spatial data communication manager.

3.6.1. Session Management Module. The session management module creates, maintains, and deletes sessions between the server PC and the sensor nodes and between the sensor nodes within a wireless sensor network. A session is created when a new spatial query is started and if an existing spatial query is finished, the corresponding session is deleted for the flexible management of sessions. In addition, the module resets the effective duration of a session when the query execution cycle is set.

3.6.2. Spatial Data Transmit/Receive Module. The spatial data transmit/receive module transmits and receives data between the server PC and the sensor nodes and between the sensor nodes within a wireless sensor network. The spatial data transmit/receive module on the server PC sends spatial attribute and spatial operator information received from the spatial interface manager to each sensor node in the wireless sensor network and again sends the final results of a spatial query received from the wireless sensor network to the spatial interface manager. The spatial data transmit/receive module

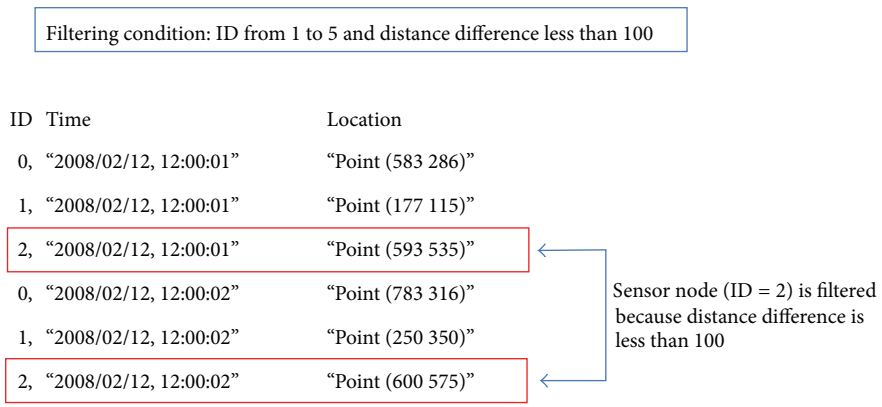

Figure 3: Example of spatial data filtering.

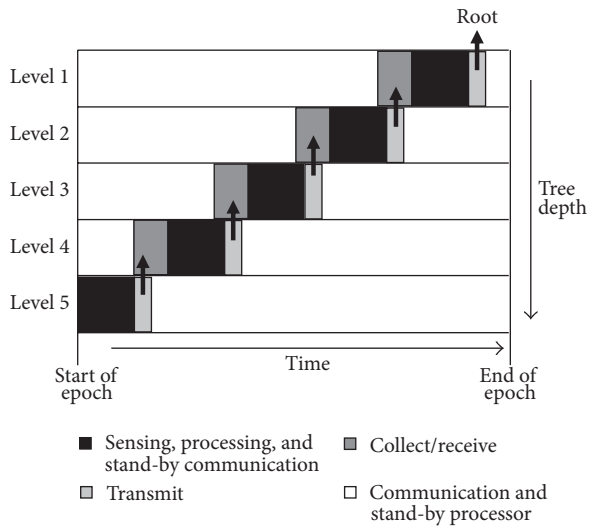

Figure 4: Spatial data transmit/receive process.

on a sensor node sends spatial data received from other sensor nodes to the query processing manager and also sends spatial data sensed by the node to other sensor nodes.

In particular, when spatial data that is to be transmitted exceed the size limit, the module divides the spatial data into smaller pieces before sending. Further, when spatial data sending fails, the module resends the data automatically. Furthermore, the module uses efficient spatial data transmit/receive methods suitable for the limited resources of sensor nodes. Figure 4 shows the spatial data transmit/receive process.

As illustrated in Figure 4, sensor nodes stay mainly in the waiting mode in order to conserve power. In addition, when a sensor node has completed operations such as data sensing, transmitting, and receiving within a certain amount of time, it is switched to the waiting mode automatically.

4. Performance Evaluation

In this section, we analyze the results of performance evaluation on the execution time, accuracy, and memory use of Spatial TinyDB proposed in this paper and TinyDB. Especially, the performance of Spatial TinyDB is evaluated both with and without spatial data filtering.

Table 10: Parameter values and query examples for performance evaluation.

Parameters	Values	
Spatial query execution cycle	256 ms	
Size limit of data	25 Byte	
The number of sensor nodes	1000, 2000, 3000, 4000	
The number of spatial queries	120	
Filtering conditions	Distance difference less than 100	
	TinyDB	Spatial TinyDB
Query examples	SELECT nodeid, temp, x, y FROM sensors WHERE temp > 40 AND x > 0 AND x < 400 AND y > 0 AND y < 400	SELECT nodeid, temp, Loc FROM sensors WHERE temp > 40 AND Contains (Polygon (0 0, 0 400, 400 400, 400 0, 0 0), Loc)

4.1. Performance Evaluation Environment. The proposed Spatial TinyDB was implemented using TinyOS 1.1.15 under Cygwin 2.5.7 as the operating system, and the development tools were nesC 1.2.8 and g++ 3.4.3 provided in TinyOS. In addition, the Java GUI was based on the Microsoft Windows XP Professional environment, and JAVA 1.4 was used as the development tool.

For the performance evaluation, we set parameters such as spatial query execution cycle, size limit of data in each transmission, number of sensor nodes, number of spatial queries executed simultaneously, and filtering condition. Table 10 lists the parameters and values set for the performance evaluation, along with the example queries used in the performance evaluation.

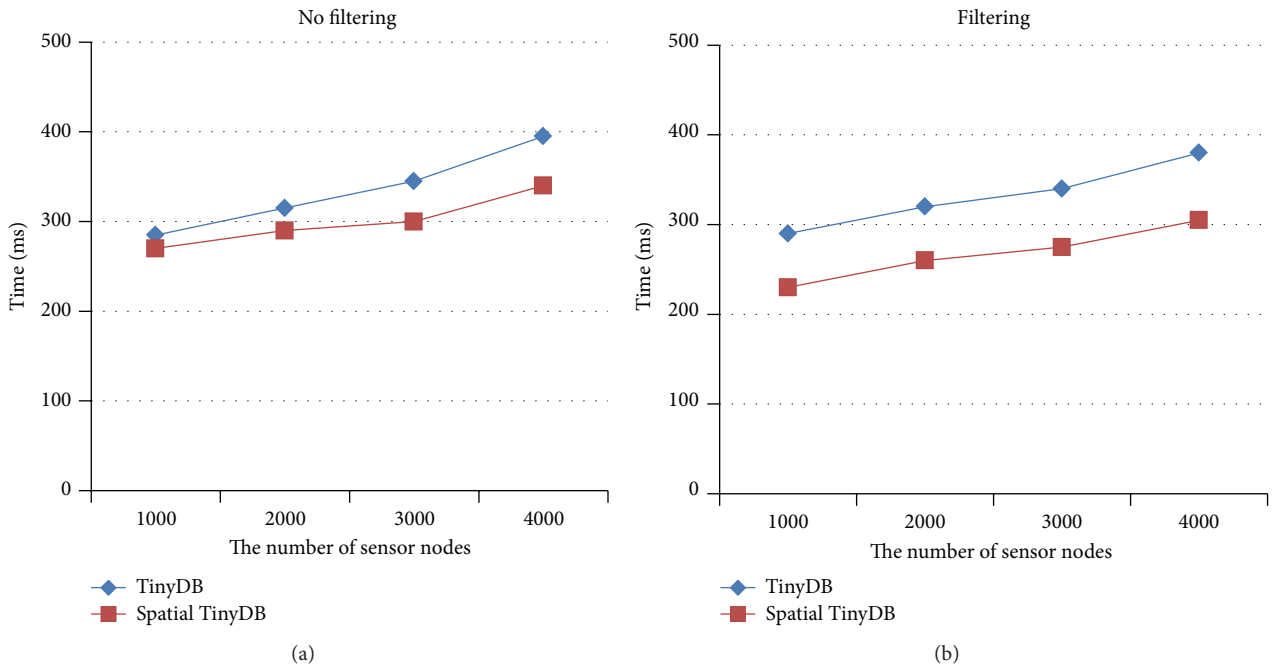

Figure 5: Measured results of the execution time.

4.2. Execution Time. Figure 5 graphically illustrates the measured results of the execution time of Spatial TinyDB and TinyDB versus the number of sensor nodes.

As illustrated in Figure 5(a) (no filtering used), Spatial TinyDB executed approximately 12% faster than TinyDB on average. This resulted from the fact that our Spatial TinyDB processes spatial queries faster because it supports spatial operators. When filtering was used (Figure 5(b)), the execution time of Spatial TinyDB was 21% faster than that of TinyDB on average. This resulted from the fact that the system load was reduced as input data were filtered before execution in Spatial TinyDB.

4.3. Accuracy. Figure 6 shows the measured results of the accuracy of Spatial TinyDB and TinyDB versus the number of sensor nodes.

As depicted in Figure 6(a) (no filtering used), Spatial TinyDB showed the same accuracy as TinyDB since there was no loss of input data in either of the two cases. In Figure 6(b), however, when filtering was used, Spatial TinyDB was approximately 7% less accurate than TinyDB on average. This resulted from the fact that input data filtering causes loss of some data in Spatial TinyDB.

4.4. Memory Usage. Figure 7 shows the measured results of the memory usage in Spatial TinyDB and TinyDB versus the number of sensor nodes.

As illustrated in Figure 7(a) (no filtering used), Spatial TinyDB showed memory usage of approximately 6% less than TinyDB on average. This resulted from the fact that memory is shared by multiple spatial queries in Spatial TinyDB. In addition, in Figure 7(b), when filtering was used, Spatial TinyDB showed memory usage approximately 11% less than that of TinyDB on average. This resulted from the fact that the spatial data stream manager reduced the volume of the input data stream not only through memory sharing but also through input data filtering.

5. Conclusions

In this paper, we propose Spatial TinyDB which is a spatial sensor database system that extends TinyDB to facilitate the efficient management of spatial sensor data in USN environments. The proposed Spatial TinyDB supports spatial data types, spatial relation operators, spatial analysis operators, and spatial trajectory operators that are in compliance with international standards. In addition, Spatial TinyDB provides memory management functions to facilitate sharing of necessary data among various spatial queries, and filtering functions to solve the overload problem by reducing the volume of the input data stream while minimizing loss of accuracy.

The results of performance evaluations indicate that Spatial TinyDB is superior to TinyDB in terms of execution time and memory use but shows slightly lower performance than TinyDB in terms of accuracy when filtering is used.

Acknowledgments

This work (Grants no. C0027296) was supported by Business for Cooperative R&D between Industry, Academy, and Research Institute funded Korea Small and Medium Business Administration in 2012.

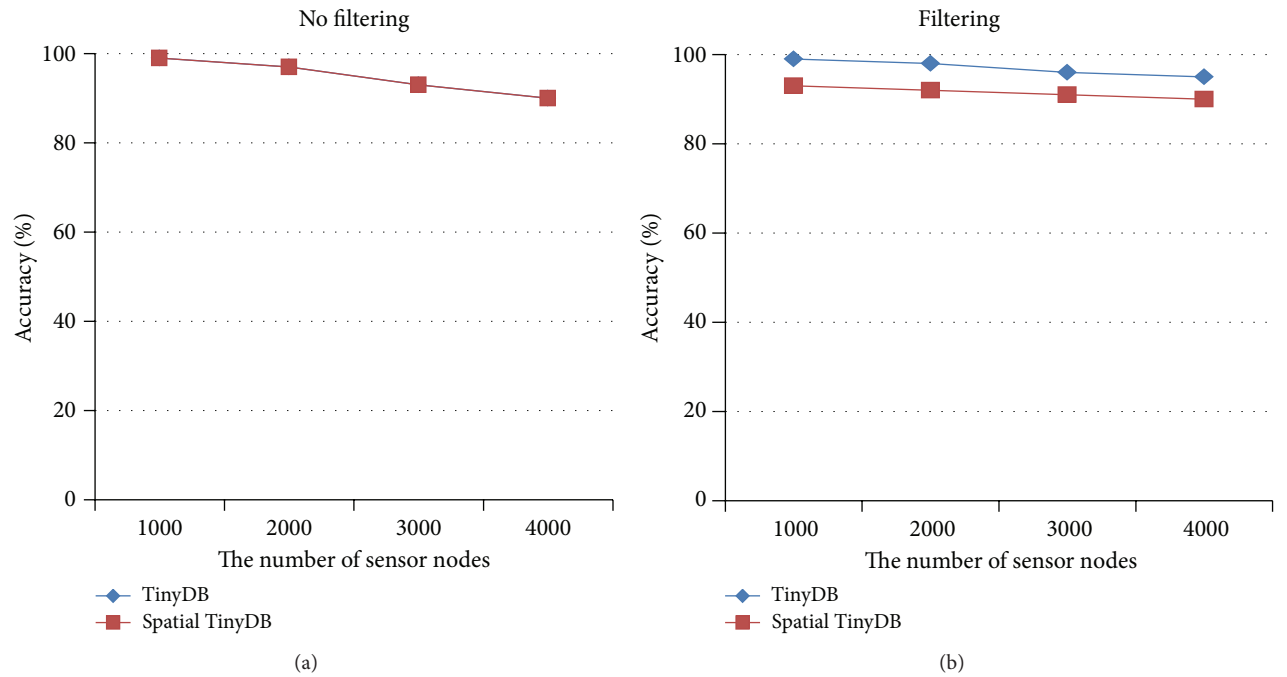

Figure 6: Measured results of the accuracy.

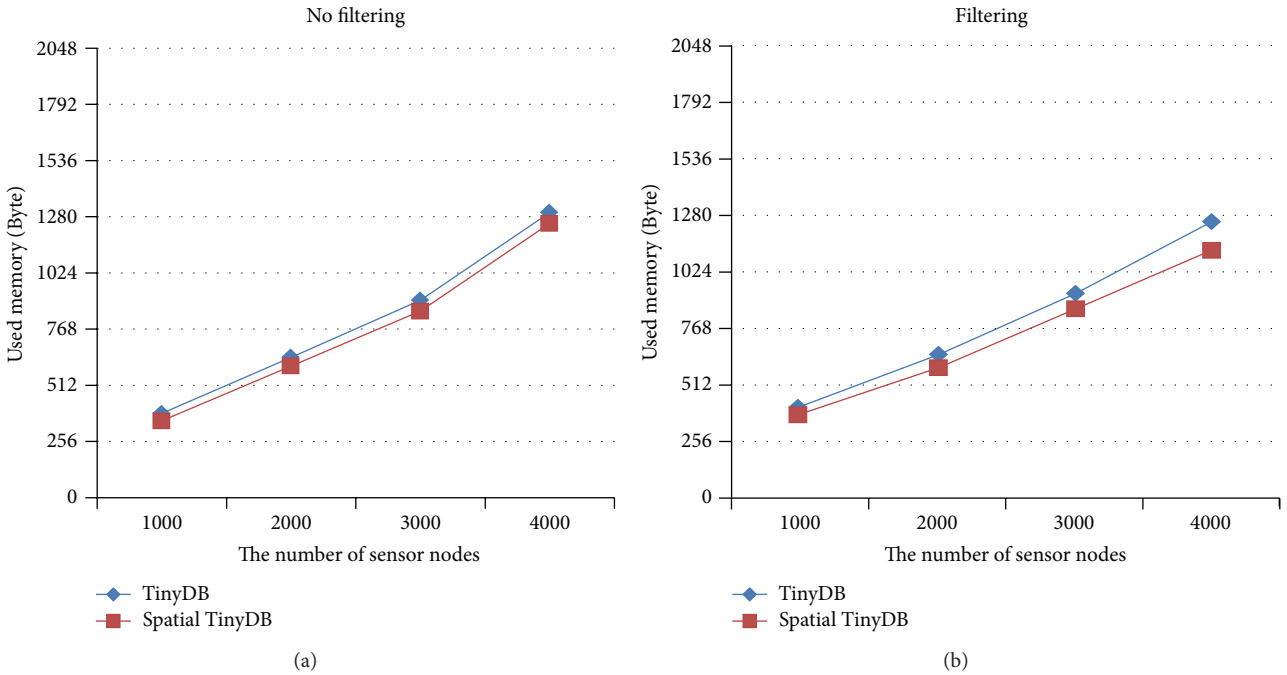

Figure 7: Measured results of the memory usage.

References

[1] M. Inoue, "A model and system architecture for ubiquitous sensor network businesses," in *Proceedings of the ITU-T Kaleidoscope Academic Conference on Innovations for Digital Inclusion*, pp. 1–8, September 2009.

[2] P. Andreou, D. Zeinalipour-Yazti, A. Pamboris, P. K. Chrysanthis, and G. Samaras, "Optimized query routing trees for wireless sensor networks," *Information Systems*, vol. 36, no. 2, pp. 267–291, 2011.

[3] E. Taslidere, F. S. Cohen, and F. K. Reisman, "Wireless sensor networks-a hands-on modular experiments platform for enhanced pedagogical learning," *IEEE Transactions on Education*, vol. 54, no. 1, pp. 24–33, 2011.

[4] S. Nittel, A. Labrinidis, and A. Stefanidis, "Introduction to advances in geosensor networks," in *GeoSensor Networks*, pp. 1–6, 2008.

[5] J. Park, K. Kim, S. Ahn, and B. Hong, "Continuous query processing on combined data stream: sensor, location and

identification," in *Proceedings of the 7th International Conference on Information Technology*, pp. 518–522, April 2010.

[6] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TinyDB: an acquisitional query processing system for sensor networks," *ACM Transactions on Database Systems*, vol. 30, no. 1, pp. 122–173, 2005.

[7] Y. Yao and J. Gehrke, "The cougar approach to in-network query processing in sensor networks," *SIGMOD Record*, vol. 31, no. 3, pp. 9–18, 2002.

[8] P. D. Felice, M. Lanni, and L. Pomante, "Design and evaluation of a spatial extension of TinyDB for wireless sensor networks," *International Journal of Computers and Their Applications Manuscript*, vol. 17, pp. 172–193, 2010.

[9] Open Geospatial Consortium, OpenGIS Implementation Specification for Geographic Information-Simple Feature Access-Part 1: Common Architecture, Version 1.2.1, 2010.

[10] P. Levis and H. Wei, "TinyDB: design, code and implementation," 2006, http://csl.stanford.edu/~pal/pubs/tinyos-programming.pdf.

[11] P. Levis, S. Madden, J. Polastre et al., "TinyOS: an operating system for wireless sensor networks," in *Ambient Intelligence*, pp. 115–148, 2005.

[12] M. Erwig and M. Schneider, "Developments in spatio-temporal query languages," in *Proceedings of the 10th International Workshop on Database and Expert Systems Applications*, pp. 441–449, 1999.

[13] J. H. Lee, K. H. An, and J. H. Park, "Design of query language for location-based services," in *Web and Wireless Geographical Information Systems*, vol. 3833 of *Lecture Notes in Computer Science*, pp. 11–18, 2005.

[14] D. Pfoser, C. S. Jensen, and Y. Theodoridis, "Novel approaches in query processing for moving objects," in *Proceedings of the 26th International Conference on Very Large Data Bases*, pp. 395–406, 2000.

[15] K. Križanović, Z. Galić, and M. Baranović, "Spatio-temporal data streams: an approach to managing moving objects," in *Proceedings of the 33rd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO '10)*, pp. 744–749, May 2010.

[16] D. Maier, P. A. Tucker, and M. Garofalakis, "Filtering, punctuation, windows and synopses," in *Stream Data Management*, chapter 3, pp. 35–58, 2005.