International Journal of Control and Automation
Vol. 11, No. 9 (2018), pp.109-118
http//dx.doi.org/10.14257/ijca.2018.11.9.11

Sensor Query Control for IoT Data Monitoring

Siwoo Byun

[1]*Dept. of Software, Anyang University*
708-113, Manan-gu, Anyang-shi, Kyonggi-do 430-714, South Korea
[1]*swbyun@anayng.ac.kr*

Abstract

IoT network consists of a large number of tiny devices that combine sensing capabilities such as light, temperature, or seismic sensors with computation and networking capabilities. In this research, a brief overview of IoT networks is presented and a new transaction management scheme for sensor databases is proposed. This management scheme is based on two-phase locking protocol using last certification reading to improve sensor query performance without sacrificing serializability.

In addition, a simulation system using CSIM API is developed. Based on the performance test, it is concluded that proposed transaction management scheme outperforms the traditional two-phase locking scheme.

Keywords*: USN, embedded device, sensor database, CSIM simulation*

1. Introduction

Sensor database is an integral component of the increasing reality of the *Internet of Things* (IoT) environment. Much of the data transmitted is small sensor data. The huge volume of data produced and transmitted frequently from sensing devices can provide a lot of information but is often considered the next big data challenge for smart IoT businesses. Recently, various sensor database systems, including TinyDB and Cougar, have been proposed for the sensor data management in IoT environment [1].

IoT server monitors the physical world by querying and analyzing sensor data. Typically, monitoring applications involve a combination of stored data (a list of sensors and their related attributes, such as their location) and sensor data. We call *sensor database* the combination of stored data and sensor data [2].

The data transaction in the sensor database server must always be consistent and reliable. Reliable transaction management consists of concurrency control and recovery management both of which are functions of the DBMS. Concurrency control ensures that the database is consistent in the event of multiple access to the database. Recovery management ensures that the database returns to a consistent or correct state after a hardware or software failure. A database transaction is a set of operations. The transaction may be an entire program or a single command. In general, concurrency management allows users to think that the database is a single-user system when in fact there are actually many simultaneous users [3].

2. Background

2.1. Sensor Network and Database

In a *ubiquitous sensor network* (USN), a number of *gateway devices* are connected to components outside of the sensor nodes, and all the communication with users goes via base stations or the gateway devices[1]. Sensor devices consist of processing, storage,

Received (June 5, 2018), Review Result (August 28, 2018), Accepted (September 6, 2018)

ISSN: 2005-4297 IJCA

sensing and networking modules. However, they have the following physical resource constraints.[5]

- **Network communication**: The wireless bandwidth of sensor network links is usually limited. In addition, the wireless network connecting the sensor nodes provides only very limited quality of service, has latency of high variance, and drops packets frequently.
- **Energy consumption**: The nodes have limited power of battery, and energy consumption is one of the main system design considerations. Small batteries provide about 3000mAh of capacity, powering the sensor node for approximately six months.
- **Computation Power**: The nodes have limited memory sizes and computing power.
- **Uncertainty in sensor readings**: Signals detected at physical sensors have uncertainty due to limitations of the sensor, and they may contain environmental noise.

That is, the sensor node has limitations with regard to the space available to store its sensed data, the capacity to process the sensed data, and the amount of electric power available. Therefore, the sensor database system, which is based on the sensor network, must be able to effectively overcome its constraints.

The example of tiny sensor node is illustrated in Figure 1. The minimal hardware of this sensor node based on TinyOS with 4Mhz, 8 bit RISC, 8 KB Main Memory, 1GB Flash Memory, and AA battery pack.

Figure 1. Example of Tiny Sensor Node

2.2. Data and Transaction Management for Sensor Databases

USN applications can monitor the physical world by remote measuring and analyzing the sensor data. The sensor data table is unbounded, and sensor database management system basically uses SQL-like queries in the form of SELECT-FROM-WHERE[4]. A typical sensor database transaction that is used in a fire monitoring server is as follows.

Transaction T : Insert measured data into sensor database after gathering temperatures in a disaster area(SN_1,...,SN_n).

TRANSACTION_BEGIN:

DO_Parallel in (SN_1,...,SN_n) {

$d_1 \leftarrow$ RemoteQuery("SELECT Get_Temperature() FROM SensorNode_1");

d_2 ← RemoteQuery("SELECT Get_Temperature() FROM SensorNode_2");

…

d_n ← RemoteQuery("SELECT Get_Temperature() FROM SensorNode_n");

}

INSERT INTO *SDB.Temperature_table* VALUES (current_time, d_1, d_2, ⋯, d_n);

COMMIT;

TRANSACTION_END:

Transaction manager of sensor database is responsible for the remote queries and the insert operation associated with the transaction denoted *T*. Upon the receipt of query request message, each sensor node performs Get_Temperature() function and returns the result messages back to the transaction manager. Upon the receipt of all query results, transaction manager stores the array of measured values into *SDB.Temperature_table* and then commits the transaction *T*.

In general, the objective of transaction concurrency control is to maximize system throughput while preventing interference among multiple requests. Transaction throughput is the number of transactions processed per unit of time in DBMS. For this concurrency control, most DBMSs use locks and *two-phase locking* (2PL) protocol [5,6]. In 2PL, locks prevent other requests from accessing a database item in use (Table 1). A database item can be a column, row, or even an entire table.

Table 1. Lock Compatibility in 2PL Protocol

Holder / Requester	Read-LCK	Write-LCK
Read-LCK	Yes	No
Write-LCK	No	No

In 2PL, a transaction must acquire a lock before accessing a database item. The concurrency control manager ensures that all transaction follow the 2PL protocol. In following the rules of this protocol every transaction is divided into two phases. First phase is a *growing phase* and acquires all the required locks. A *shrinking phase* follows in which it releases all the locks acquired and cannot request for any more locks. The transaction need not request for all the locks at the same time. The transaction would usually acquire some locks, does some processing and continues to get more locks as needed. But the transaction would only release all the locks when no new locks are needed[3].

In optimistic approach transactions can access the database without obtaining locks. The concurrency control manager then checks for conflicts, if a conflict has occurred the concurrency manager performs a rollback operation and restarts the problematic transaction [5].

As compared to traditional DBMS which processes data already in the system, acquisition DBMS such as the monitoring server generates the data in the sensor network. Therefore, most monitoring queries have the characteristic of long-running transaction due to the long delay of data measuring and the large amount of network communication between monitoring server and sensors. In this respect, it should enable the users to effectively handle monitoring sensor database which is associated with a number of sensor nodes while overcoming its constraints.

3. Proposed Transaction Control Approach

As compared to general databases with wired networks, sensor database systems have inferior features of slow wireless networks and long measuring time for search or update queries. Therefore, in order to achieve high database performance, a new transaction management scheme that is able to efficiently handle the characteristics of the sensor databases must be devised. In this respect, a new transaction management scheme called *Sensor Network Transaction Management*(SNetTM) is hereby proposed. The main idea behind SNetTM is based on 2PL and *Last Certification Reading*.

3.1. Notion of *Last Certification Reading*

Transaction management is the activity of coordinating actions of processes when two or more transactions are executed concurrently. In order to preserve database consistency, most transaction management schemes guarantee *serializability* by adopting 2PL mechanism which is the most basic type in disk-based database system and main memory-based systems.

Any schedule generated by 2PL algorithm is serializable[5]. Among the variants of 2PL, The *Strict Two-phase Locking Transaction Manager*(STPL-TM) is considered to be the most well-known transaction manager for practical database systems. SSTPL-TM removes all the locks if the transaction ends. More specifically, a transaction's locks are released after data manager acknowledges the processing of commit or abort. Although STPL-TM scheme is efficient and guarantees serializable execution in general database environments, STPL-TM needs to be improved in order to achieve high database performance in a slow sensor database environment. Thus, we devise the notion of the *Last Certification Reading* (LCR). In general, maintaining another version of data potentially increases parallelism without sacrificing serializability when processing transactions [7,8]. Thus, LCR can achieve high transaction performance by allowing last version reads and efficiently handling slow operations in lock management processes.

3.2. Conflict Control Method of *SNetTM*

In order to handle locks in SNetTM, it is required to devise a new lock types called *notice* and *certification* for secure execution of a write transaction and a lock operation manager. Regarding lock types, four kinds of locks are used such as *read, notice, write, and certification.* These four lock modes will be referred hereafter as *Read-LCK, Noti-LCK, Write-LCK*, and *Certi-LCK* for short, respectively(Table 2). The Read-LCK and Write-LCK are identical to read lock and write lock respectively in general concurrency control schemes.

Table 2. Lock Compatibility in SNetTM

Holder / Requester	Read-LCK	Noti-LCK	Write-LCK	Certi-LCK
Read-LCK	Yes	Yes	Yes	No
Noti-LCK	Yes	No	No	No
Write-LCK	Yes	No	No	No
Certi-LCK	No	No	No	No

In case of write operations, sensor nodes of write group should notify their aliveness before they can be written. The notice operation is very slow operation which takes a few milliseconds. The actual write operation is also slow operation which handles a number of sensor nodes over slow wireless network, as compared to general databases over fast wired network. Therefore, in proportion to the ratio of write transaction, transaction

throughputs and responsiveness could be considerably degraded due to high probability of conflict occurrences and long lock-holding time. Furthermore, the recovery cost of write transactions is considered to be far more significant due to their long operation time. Therefore, in order to guarantee secure execution of long-time write transactions, it is necessary to confirm that sensor nodes associated with the write transactions are ready to update prior to the actual write operations. This can be achieved by employing Noti-LCK in order to later enforce the actual write operations in the sensor nodes. Noti-LCK locks the segment blocks associated with the write transactions at sensor node reservation phase. By confirming the receipt of granted Noti-LCK messages, the Noti-LCK is upgraded to Write-LCK and then the actual write operations are enforced in the segment blocks.

In STPL-TM, a write lock on a data item x, denoted by Write-LCK(x), prevents transactions from obtaining read locks on x, denoted by Read-LCK(x). In order to avoid this lock conflict which degrades transaction performance, LCR exploits alternative version of x. When a transaction, Tr, writes into x, it creates a new version x_i of x. Other transactions are allowed to read the alternative version of x. Therefore, read operations on x are not delayed by a concurrent write operation of x. There is also certification activity involved. For this version coordination, data manager(DM) should maintain two versions of each data item and only one of those versions was written by a committed transaction. Once a transaction Tr that wrote x commits, the previous committed version of x becomes inaccessible.

SNetTM scheduler which exploits notion of LCR set Read-LCK, Noti-LCK, and Write-LCK at the usual time, when it processes read or write operations. When SNetTM finds that a user request is about to commit, it converts all Write-LCKs to Certi-LCKs. When SNetTM scheduler receives a write operation, denoted by $w(x)$, it attempts to set Noti-LCK(x) and then Write-LCK(x). Since a write lock conflicts with a certification lock and with each other, SNetTM scheduler delays Write-LCK(x) if another transaction already has a Write-LCK(x) or Certi-LCK(x). Otherwise, SNetTM set Write-LCK(x), translates $w(x)$ into $w(x_i)$, and sends $w(x_i)$ to the DM. When SNetTM scheduler receives a read operation, denoted by $r(x)$, it attempts to set Read-LCK(x). Since Read-LCK only conflicts with Certi-LCK, it can set Read-LCK(x) as long as no transaction already owns a Certi-LCK(x). If a transaction Tr already owns Write-LCK(x) and has therefore written x_i, then SNetTM scheduler translates $r(x)$ into $r(x_i)$, and sends $r(x_i)$ to the DM. Otherwise, it waits until it can set Read-LCK(x), and then set Read-LCK(x), translates $r(x)$ into $r(x_j)$, where x_j is the alternative version of x which is the most recently committed, and sends $r(x_j)$ to the DM.

If SNetTM scheduler receives a commit operation, c, indicating that transaction Tr has terminated, it attempts to convert Tr's Write-LCK into Certi-LCK. On those data items where such Read-LCKs exist, the SNetTM delays the lock conversion until all Read-LCKs are released. Therefore, the effect of Certi-LCK is to delay Tr's commitment until there are no active read operations of data items it is about to overwrite.

3.3. Correctness of *SNetTM*

The correctness of SNetTM is proven by using multiversion serializability and confirming that all histories produced by SNetTM are 1SR(*one copy serializable*). The interested reader is directed to the theory of multiversion serializability. To list the properties of histories produced by SNetTM scheduler, we need to include the operation f_i, denoting the certification of T_i. Let H be a history over $\{T_0, T_1, \ldots T_n\}$ produced by SNetTM scheduler. Then H must satisfy the following properties.

P1: for every T_i, f_i follows all of T_i's reads and writes and precedes T_i's commitment.

P2: for every $r_k(x_j)$ in H, if $j \neq k$, then $c_j < r_k(x_j)$; otherwise $w_k(x_k) < r_k(x_k)$.

P3: for every $w_k(x_k)$ and $r_k(x_j)$ in H, if $w_k(x_k) < r_k(x_j)$, then $j=k$.

P4: if $r_k(x_j)$ and $w_i(x_i)$ are in H, then either $f_i < r_k(x_j)$ or $r_k(x_j) < f_i$.

P5: for every $r_k(x_j)$ and $w_i(x_i)$ (i, j, and k distinct), if $f_i < r_k(x_j)$, then $f_i < f_j$.

P6: for every $r_k(x_j)$ and $w_i(x_i)$, $i \neq j$ and $i \neq k$, if $r_k(x_j) < f_i$, then $f_k < f_i$.

P7: for every $w_i(x_i)$ and $w_j(x_j)$, either $f_i < f_j$ or $f_j < f_i$.

P2 says that every $r_k(x_j)$ either reads a certified version or reads a version written by itself. *P3* says that if T_k wrote x before the scheduler received $r_k(x)$, then it translates $r_k(x)$ into $r_k(x_k)$. *P4* says that $r_k(x_j)$ is strictly ordered with respect to the certification operation of every transaction that writes x. This is because each transaction T_i that writes x must obtain a certification lock on x. For each transaction T_k that reads x, either T_i must delay its certification until T_k has been certified, or else T_k must wait for T_i to be certified before it can set its read lock on x and therefore read x. *P5*, combined with *P2*, says that each $r_k(x_j)$ either reads a version written by T_k or reads the most recently certified version of x. *P6* says that a transaction T_i that writes x cannot be certified until all transactions that previously read a version of x have been certified. This follows from the fact that certification locks conflict with read locks. *P7* says that the certification of every two transactions that write the same data item are atomic with respect to each other.

Theorem 1: Every history H produced by a *SNetTM* scheduler is *1SR*.

Proof: By *P1*, *P2*, and *P3*, H preserves reflexive reads-from relationships and is recoverable, and therefore is a multi version history. Define a version order $\ll$ by $x_i \ll x_j$ only if $f_i < f_j$. By *P7*, $\ll$ is indeed a version order. We will prove that all edges are in certification order. That is, if $T_i \rightarrow T_j$, then $f_i < f_j$. This edge corresponds to a reads-from relationship, such as T_j reads x from T_i. By *P2*, $f_i < r_k(x_i)$. By *P1*, $r_j(x_i) < f_j$. Hence, $f_i < f_j$.

Consider a version order edge induced by $w_i(x_i)$, $w_j(x_j)$, and $r_k(x_j)$ (i, j, and k distinct). There are two cases: $x_i \ll x_j$ and $x_j \ll x_i$. If $x_i \ll x_j$, then the version order edge is $T_i \rightarrow T_j$, and $f_i < f_j$ follows directly from the definition of $\ll$. If $x_j \ll x_i$, then the version order edge is $T_k \rightarrow T_i$. Since $x_j \ll x_i$, $f_j < f_i$. By *P4*, either $f_i < r_k(x_j)$ or $r_k(x_j) < f_i$. In the former case, *P5* implies $f_i < f_j$, contradicting $f_i < f_j$. Thus $r_k(x_j) < f_i$, and by *P6*, fk < f_i as desired. This proves that all edges are in certification order. Since certification order is embedded in a history, which is acyclic by definition, serialization graph in multi version history is acyclic too. Thus, H is *1SR*.

4. Performance Experiment

4.1. Experiment System Setup

The performance of SNetTM is compared to the well-known transaction management scheme, STPL-TM, by means of computer simulation. The queuing model used in the experiment is a closed queuing model for a sensor database.

The system model for the simulation comprises four distinct components: *sensor database transaction generator* (SdbTG), *sensor database transaction manager* (SdbTM), *sensor database lock manager* (SdbLM), and *sensor database data manager* (SdbDM). SdbTG is responsible for generating user transactions, which is a sequence of transactions. SdbTM manages the user transaction from beginning to commitment. SdbTM analyzes the user transaction and sends it to scheduling request queue of SdbLM. SdbLM accepts the request one by one in FCFS manner and processes it according to scheduling algorithms such as SNetTM and STPL-TM. SdbLM also manages the locks required for the transactions. The number of sensor nodes, *num_SNodes*, effectively

controls the concurrency level of the system. Central to our simulation is the closed queuing model shown in Figure 2.

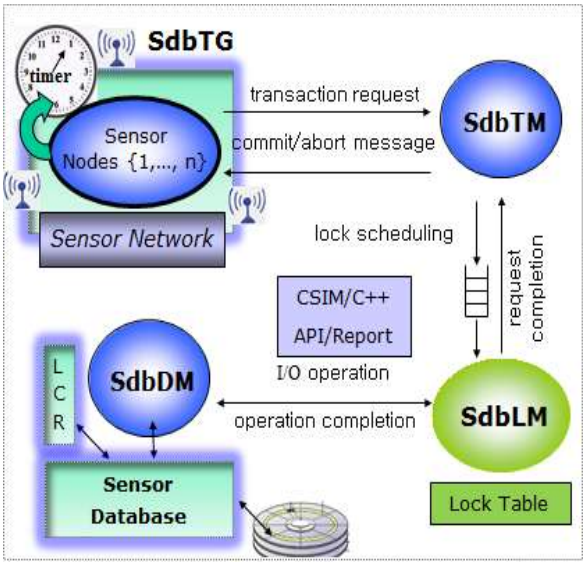

Figure 2. Queuing System Model for Simulation

The simulation model was programmed using the CSIM [9,10] discrete-event simulation software, and our experiments run using the Intel Core TM i7 CPU (@ 2.80 GHZ) with 16 GB of RAM.

4.2. Simulation Parameters and Performance Indices

The major simulation parameters are shown in Table 3. Since it is thought that the level of contention in sensor database is most important factors on the transaction throughput and the abort ratio, varied update ratio evaluates the performance under various levels of write operation. Other values to be reasonable for comparing SNetTM with STPL-TM were also determined while keeping simulation times reasonable.

Table 3. Major Simulation Parameters and Setting

System Parameters	Description	Value
num_SNodes	Number of sensor nodes in a sensor group	300-1,100 in steps of 100
Sdata_ReadDelay	Access delay to read an object in a sensor	36 msecs
Sdata_WriteDelay	Access delay to write an object in a sensor	266 msecs
Sdata_TransDelay	Transmission time between sensor node and sensor database server	0.1~2 msecs random
Sdata_UpdateRatio	Probability of update operation	20~80 % in steps of 10

Our primary metrics are *transaction throughput rate* and *transaction abort ratio*. A lower abort ratio corresponds to an enhanced degree of transaction concurrency. An additional performance-related metric is average elapsed time.

4.3. Performance Study

The analysis of the simulation results of the two transaction management schemes is hereby presented in this section. First simulation is used to test the effect of concurrency level on the performance of transaction management schemes, *num_SNodes* varies. The update ratio is set to a default value of 25 percent. The overall throughput is presented in Figure 3, and its corresponding average elapsed time is presented in Figure 4. The transaction abort ratio is depicted in Figure 5. The average elapsed time increase with the concurrency level. In this experiment, it is observed that the highest throughput is exhibited by SNetTM, followed by STPL-TM.

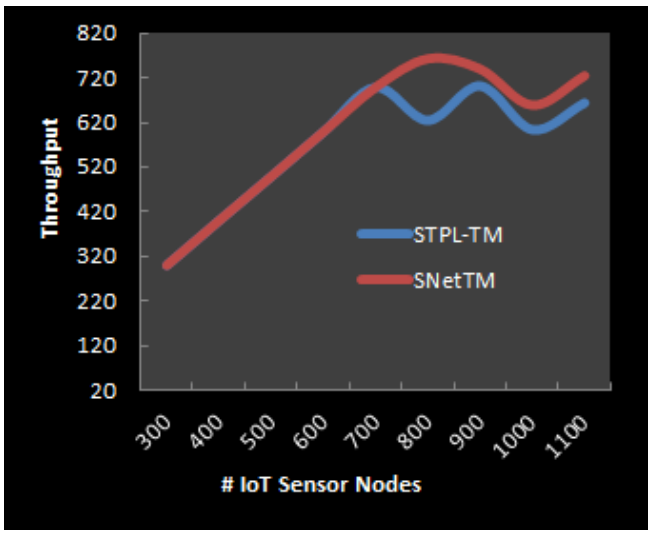

Figure 3. Average Transaction Throughput

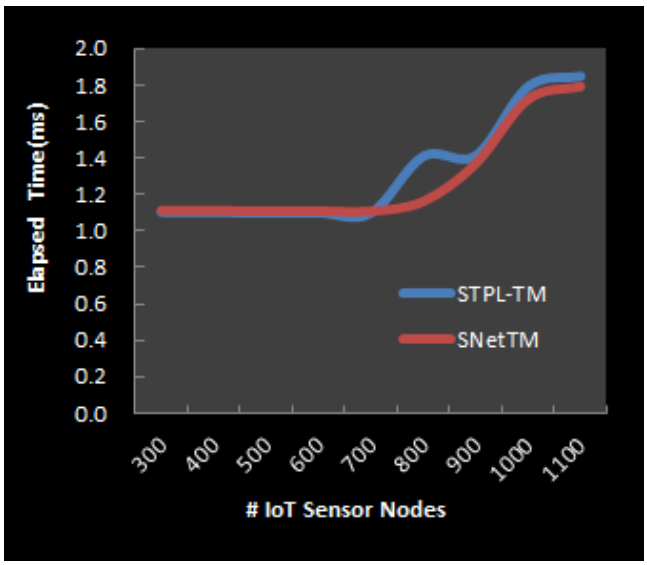

Figure 4. Average Transaction Elapsed Time (Milliseconds)

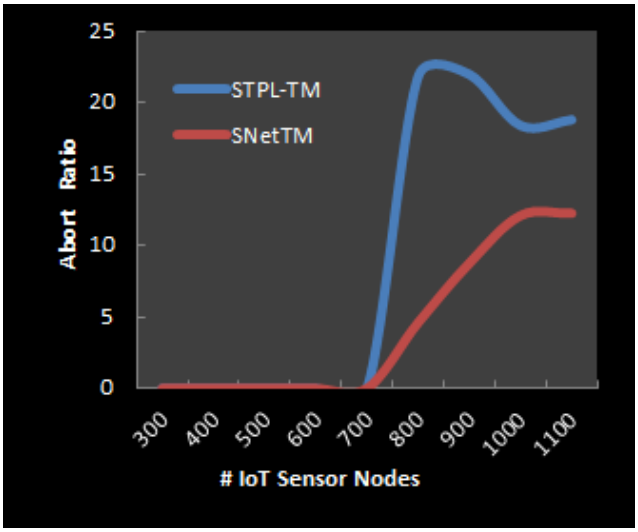

Figure 5. Average Transaction Abort Ratio

In Figure 3, it is observed that the throughput of the two scheme begins to be decreased beyond 700. This fact implies that adding more transactions beyond that range simply contributes to increasing resource contention. That is, the performances of the two transaction management schemes are mainly limited by the factor of data contention such as read-write and write-write operation conflicts.

In Figure 3, it is observed that the performance gain of SNetTM relative to STPL-TM begins to decrease as *num_SNodes* increases beyond 700, although SNetTM achieves higher performance than STPL-TM by reducing transaction aborts caused by excessive read-write operation conflicts throughout the whole range of num_SNodes. When *num_SNodes* reaches the highest data contention point of 800, the transaction throughput of SNetTM begins to be lowered. This implies that SNetTM also experiences the negative effect of data contention with the same degree as in STPL-TM under the high data contention environment. In contrast, under the middle and high data contention environment of 600-to-1100, SNetTM shows much higher transaction throughput than STPL-TM.

In Figure 4, it is observed that the elapsed time of SNetTM and STPL-TM increases gradually, as the concurrency level increase. At the overall concurrency levels, the elapsed time curve of SNetTM is superior to that of STPL-TM. The performance gain of SNetTM relative to STPL-TM reaches 22 percent when 800 *num_SNodes* exist. This means that the gain in SNetTM over STPL-TM comes from the implementation of notion of last certification reading and notice lock management. However, STPL-TM, inevitably aborts most transactions which is about 21.8 percent due to excessive read-write and write-write operation conflicts. As compared to STPL-TM, SNetTM successfully overcomes the negative effect of the excessive conflicts by employing alternative version read and exploiting notice write lock under the same condition, and thus successfully coordinates read/write transactions.

5. Conclusions

It is therefore proposed that this new transaction management scheme named *SNetTM* be adopted in order to achieve high transaction performance in sensor database systems. Unlike the previous approaches, SNetTM allows certified version reads and efficiently handles slow query operations in lock management processes. A simulation model based on closed queuing system to show the performance of SNetTM is also hereby proposed. This simulation results show that SNetTM outperforms the traditional STPL-TM scheme in terms of abort ratio and throughput. This is because SNetTM successfully overcomes

the negative effect of the excessive conflicts of read and write operations by employing certified version coordination and exploiting notice write lock under the same condition. Since SNetTM has a generic functionality of efficient transaction management in sensor database environments, it can be widely employed in ubiquitous sensor networks.

Acknowledgments

This research was supported by Basic Science Research Program funded by the Ministry of Education, Science and Technology(2018R1D1A1B07044418, NRF-2015R1D1A1A01057675).

References

[1] K. Dong-Oh, L. Lei, S. In-Su, K. Jeong-Joon and H. Ki-Joon, "Spatial TinyDB: A Spatial Sensor Database System for the USN", Int. Journal of Distributed Sensor Networks, **(2013)**, pp. 1-10.
[2] P. Bonnet, J. Gehrke and P. Seshadri, "Towards Sensor Database Systems", Proceedings of the Second International Conference on Mobile Data Management, Hong Kong, **(2011)** January, pp. 3-14.
[3] http://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0021_53_database_system CMDB5103_ Database_System_ 09.pdf, **(2017)**.
[4] C. Yong Yao and Johannes Gehrke, "Cougar Approach to In-Network Query Processing in Sensor Networks", ACM SIGMOD Record, vol. 31, no. 3, **(2002)** September, pp. 9-18.
[5] R. Elmasri and S. Navathe, "Fundamentals of Database System", Addison-Wesley, NY, **(2015)**.
[6] Y. Wu, J. Arulraj, J. Lin, R. Xian and A. Pavlo, "An Empirical Evaluation of InMemory MultiVersion Concurrency Control", Proceedings of the VLDB Endowment, vol. 10, no. 7, **(2017)**, pp. 781-792.
[7] S. Byun, "Design of Efficient Index Management for Column-based Big Databases", International Journal of Internet of Things and Big Data, vol. 2, no. 1, **(2017)** May.
[8] J. Lee, M. Muehle, N. May, F. Faerber, V. Sikka, H. Plattner, J. Krueger and M. Grund. "High-Performance Transaction Processing in SAP HANA", IEEE Data Eng. Bull, vol. 36, no. 2, **(2013)**.
[9] http://www.mesquite.com/documentation/documents/CSIM20_User_Guide-C++.pdf, CSIM20 Simulation Engine (C++ Version), **(2019)**.
[10] S. Byun, "Efficient Transaction Control in Sensor Data Network", International Journal of Digital Contents and Applications for Smart Devices, vol. 4, no. 2, **(2017)** December.

Author

Siwoo Byun, he received his B.S. degree in Computer Science from Yonsei University in 1989 and earned his M.S. and Ph.D. in Computer Science from Korea Advanced Institute of Science and Technology (KAIST) in 1991 and 1999. Currently, he is teaching and researching in the areas of distributed database systems, embedded systems, mobile computing, flash memory database, and fault-tolerant systems in Anyang University.

