Analysis and Comparison of Concurrency Control Protocols for Wireless Sensor Networks

Christoph Reinke, Nils Hoeller, Stefan Werner, Sven Groppe, Volker Linnemann
Institute of Information Systems - University of Luebeck
Ratzeburger Allee 160, 23538 Luebeck, Germany
reinke, hoeller, werner, groppe, linnemann (at) ifis.uni-luebeck.de

Abstract—**In recent years the sensor network databases TinyDB and StonesDB have emerged. While both provide a useful abstraction layer for querying data, live data in the case of TinyDB and historic data in the case of StonesDB, neither of these approaches provide transaction processing capabilities. Transaction processing capabilities are needed to guarantee the consistency, for instance, in the case of data updates at runtime or the sophisticated requirements of emerging wireless sensor and actor networks.**

In this paper we analyze traditional concurrency control approaches and compare them with regard to their usability in wireless sensor networks. Therefore, we implemented the traditional Strict Two Phase Locking (S2PL), Timestamp Ordering (TO) and Forward Oriented Optimistic Concurrency Control (FOCC) by validation. We show in experiments with the network simulator Shawn, that locking exhibits better commit rates and lower costs under a variety of conditions compared to timestamp ordering and validation. We also implemented locking for the sensor node platform Pacemate to show the feasibility of our approach. We believe that efficient concurrency control can broaden the application spectrum of sensor network databases and is also vital for the emerging wireless sensor and actor networks.

Index Terms—**Concurrency Control, Isolation, Wireless Sensor Networks**

Today's wireless sensor networks can consist of hundreds or thousands of tiny microcontrollers equipped with sensors and radio interfaces. The task of a sensor network is mainly to monitor the environment and to provide the observations to the user so that they can be displayed, analyzed and archived in a convenient way.

To simplify this task, a number of sensor network databases (SDBS) have emerged in recent years. The most prominent ones are TinyDB [17], SwissQM [18] and StonesDB [7]. While TinyDB is mainly an abstraction layer for simplifying the querying of wireless networks measuring live data by providing an SQL-like interface, StonesDB makes use of a node's flash memory to store historic data. In StonesDB the measured data is then processed directly on the node and only relevant items are forwarded to the gateway in order to save transmission costs.

What has been missing in sensor databases compared to traditional databases until now is the capability of transaction processing. This has mainly two reasons: On the one hand, the severe resource constraints of wireless sensor networks make the implementation of complex protocols complicated. On the other hand, until now exclusively read-only queries have been considered in sensor network databases. However, as Guergen et al. [10] point out, the variety of emerging sensor networks applications also demands for update queries, for example, system management queries which allow the alteration of a data model at runtime.

Consider as an example a continuous query that asks for the average temperature in Celsius in each section of a factory. The result is used to trigger a fire alarm if a certain threshold is met. Now consider a concurrent update query which modifies the measuring unit of section A to Fahrenheit. If no concurrency control is present which guarantees the isolation of both transactions, a false alarm might be triggered.

System management queries are not only needed for changing a measuring unit, but also, for example, when nodes are relocated and the position information needs to be changed at runtime. Another application domain for concurrency control protocols and also atomic commit protocols in wireless sensor networks are sensor and actor networks.

Whenever nodes not only measure their environment but also perform actions based on their measurements, a transaction concept might be needed, especially in a military environment, but also in civil application domains. As an example for distributed agreement, consider four unmanned vehicles crossing an intersection [2]. Somehow the cars must agree on an order for crossing the intersection to prevent accidents. In general, the coordination of multiple actors can be required in the following situations [1]:

- One actor may not be sufficient to perform the requested operation.
- It might be required that an action is performed by exactly one actor.
- If multiple actors perform an operation, this has possibly to be done simultaneously so that synchronization is required.
- If a region is covered by multiple actors, they might all have to cover their own region, so that no overlaps occur. Hence, mutual exclusion must be guaranteed.
- Ordered execution of tasks might be relevant.

According to Akyildiz et al. [1], a task performed by actors is defined as an atomic unit of computation and control. Also, it might be required that a certain task is by no means executed by all possible actors at the same time, like in the case of disposers of a tranquilizing gas, which could lead to catastrophic events.

Hence, to achieve a well defined behavior, concurrency control for wireless sensor networks is needed, which is neither considered by Ayari et al. [2] nor by Akyildiz et al. [1]. We believe that the well-known serializability concept from the database area [3] is able to fill this gap in order to enable more sophisticated sensor network applications. However, the usage of well known algorithms like two-phase locking, timestamp ordering or validation is not straight forward since wireless sensor networks pose new challenges in terms of message loss and severe resource constraints.

In this paper we analyze the mentioned concurrency control protocols with regard to their usage in wireless sensor networks and integrate an adapted version of each protocol in the well known Two Phase Commit (2PC) protocol [3]. Evaluation results of 2PC obtained from simulation and experiments with real sensor nodes can be found in [20], [21], [22], [23], [24].

A. Contributions

The remainder of this paper is structured as follows:

- In Section I, we outline related work on concurrency control in wireless sensor networks, mainly on sensor network databases and concurrency control protocols in mobile ad hoc networks.
- We describe in Section II how we adapted the traditional concurrency protocols locking, timestamp ordering and validation for wireless sensor networks with respect to their severe resource constraints.
- We evaluate and compare the implemented protocols by performing experiments with the network simulator Shawn in Section III. We also describe our implementation of locking for the sensor node platform Pacemate. To the best of our knowledge this is the first paper reporting an implementation of a database concurrency control protocol for real sensor nodes.
- Section IV concludes this paper and proposes future work.

I. Related Work

In this section we briefly outline related work in the areas of sensor network databases and concurrency control in mobile and ad hoc networks.

A. Sensor Network Databases

The existing sensor network databases TinyDB [17] and StonesDB [7] do not provide any transaction processing capabilities. Levent et al. [10] claim that these capabilities are needed and describe an algorithm for the concurrent processing of continuous queries in a wired sensor network. The algorithm is not implemented and the unique properties of wireless networks are not considered in depth by the authors. Therefore, we review related work in the area of mobile and ad hoc networks.

B. Concurrency Control in Mobile Ad Hoc Networks

There are significant differences between Mobile and Ad hoc Networks (MANets) on one hand and wireless sensor networks on the other hand. Common examples of MANet devices are mobile phones or palms. Wireless sensor networks are considered to be a subset of MANets. They often consist of a higher number of nodes which are distributed with a higher density and in varying topologies. Wireless sensor networks can also suffer from higher failure rates and also from more severe restrictions in terms of storage capacity and processing power. Another important difference is that sensor nodes are seldom recharged, in contrast to devices like mobile phones or palms. Therefore, it is much more important to save energy in wireless sensor networks. Since transmitting and receiving consumes the most power in sensor networks, most energy can be saved by reducing the transferred volume of data.

Lee et al. [15] consider MANets and introduce a protocol for broadcast environments where read only transactions can be validated on clients only. Since we do not differentiate between clients and servers, we need a more general kind of concurrency control.

Brayner et al. define the correctness criterion Mobile Semantic Serializability [6] and describe SESAMO [5], a concurrency control algorithm which relaxes global serializability and provides mobile semantic serializability instead.

Xing et al. [27] propose an optimistic concurrency control algorithm for MANets called Sequential Order with Dynamic Adjustment (SODA) which relies on a clustering algorithm. Although using clustering could also be beneficial for our implementation, the authors do not consider the severe resource constraints of wireless sensor nodes.

II. Implementing Traditional Concurrency Control Protocols for Wireless Sensor Networks

As outlined in the introduction, emerging applications for wireless sensor networks demand for concurrency control. Examples are update queries in sensor network databases like system management queries and wireless sensor and actor networks. In this section, we outline our implementation of traditional concurrency control protocols for wireless sensor networks. Every protocol was integrated in the Two Phase Commit (2PC) protocol to provide atomicity and serializability.

One major difference between traditional distributed database environments and wireless sensor network databases are the missing logging capabilities. Traditional database systems provide the possibility to recover from a crash of a participant by writing temporary data to a stable storage which can be accessed after the failed node has been restarted. We did not implement logging in our protocols since contrary to database server, crashed sensor nodes are considered to be either damaged our out of power. Hence, they do not restart and cannot recover.

Another major challenge are the severe resource constraints of sensor nodes. We describe how we have taken them into account in the next subsections.

A. Locking - Strict Two Phase Locking (2PL)

The widely used commercial Strict Two Phase Locking (S2PL) is described by Gray et al. [9]. It guarantees serializability by first acquiring locks on all accessed resources (phase 1), performing the requested operations if all locks were granted and releasing the locks afterwards (phase 2). Since our application domain is a distributed sensor network, we also implemented a distributed management of locks. This means each participating sensor node locks the accessed resources (if not already locked) when receiving a begin vote message and unlocks the resources when receiving the commit or abort decision from the coordinator. Global deadlocks due to locking generate automatically voting-deadlocks in 2PC, and are thus resolved automatically by 2PC when the respective coordinator timeout expires. This behavior was generalized by the principle of commitment ordering by Raz [19]. If a node does not receive the commit or abort message from the coordinator, the respective resource remains locked. To prevent this blocking, our participants broadcast a "help me" message if they do not receive the decision message. While this decreases the number of blocked resources, it cannot guarantee that no resources remain locked since coordinating nodes can fail and message loss can occur repeatedly.

B. Timestamp Ordering (TO)

Timestamp based concurrency control is also called Timestamp Ordering (TO) and is described by Bernstein et al. [4]. Using this technique, serializability is guaranteed by timestamps assigned to data objects and transactions. The validation of timestamps is performed at each sensor node participating in the transaction, leading to an inherently distributed, deadlock free protocol. A globally unique timestamp is assigned to every transaction at begin of transaction (BOT). A globally unique timestamp can for instance be determined by using the node ID and its local time [14]. Without synchronization, no monotonic increasing of the timestamps is guaranteed, which can lead to the abort of transactions. Using timestamp ordering, the global execution order of transaction is predefined by their timestamps. Conflicting operations must occur in the order of the transaction timestamps. Although time synchronization in wireless sensor networks is possible and a lot of publications deal with this topic (see, for instance, Römer et al. [25]), it induces a significant overhead and it is hard to guarantee globally unique timestamps in a timely manner under message loss.

To be able to validate the transaction timestamps, a read timestamp (RTS) and a write timestamp (WTS) are assigned to every data object. These timestamps are always updated if the data object is accessed by a transaction. A read transaction T on an object x is aborted, if $ts(T) < WTS(x)$ holds. A write transaction T on an object x is aborted, if $ts(T) < \max\{WTS(x), RTS(x)\}$ holds. Although timestamp ordering is deadlock free, blocking can occur. Assume T1 has successfully updated data object x, but has not committed yet. Assume also, that T2 wants to read x. If T2's read access is allowed and T1 is aborted later, then T2 must also be aborted. This can lead to cascades. So T2 must wait until T1 has committed, which leads to a blocking situation analogous to locking.

We decided to abort transactions in this case, because on the one hand, there may be timely constraints due to the application and on the other hand, because the used 2PC would abort the transaction anyway when the respective coordinator timeout expires.

C. Forward Oriented Optimistic Concurrency Control (FOCC) by Validation

Optimistic concurrency control (OCC) by validation is introduced by Kung et al. [13]. OCC protocols are based on the assumption that conflicts occur rarely, which makes locking, for instance, an unnecessary overhead. We differentiate between the three phases: read, validate and write.

i) Read phase: In this phase a transaction reads its needed data objects and performs its writes to its own private workspace.

ii) Validation phase: The validation phase is started at EOT. It is checked if conflicts between concurrent transactions have occurred. If this is the case, the transaction is aborted. Neither blocking nor deadlocks can occur, but frequent aborts can lead to starvation of a transaction.

iii) Write phase: The updates stored in the private workspace are written to the log and update the actual database, becoming visible for other transactions.

To perform the validation, the objects to be accessed by transaction T_i are assigned to its write set $WS(T_i)$ respectively read set $RS(T_i)$. According to [11], two classes of OCC protocols can be distinguished: Backward Oriented Optimistic Concurrency Control (BOCC) and Forward Oriented Optimistic Concurrency Control (FOCC). BOCC validates transactions against already committed transactions, while FOCC validates transactions against concurrently running transactions (see Listing 1). Both techniques guarantee serializability by making sure that a transaction has read all changes written by all previous successfully validated transactions.

The advantage of FOCC is that only real conflicts lead to aborts and that in mobile environments, nodes can work autonomously when temporary disconnected. Also, FOCC is more flexible. While BOCC always aborts the validating transactions in case of conflicts, FOCC also allows to abort the other conflicting transaction. When using distributed validation, every sub transaction is validated on the node it is executed on. Global transactions are synchronized by means of 2PC as follows: The prepare message is also used as a request for validation. After a successful local validation, every sub transaction saves its changes in a local workspace and sends a vote commit. If the validation fails, the sub transaction sends vote abort and deletes the workspace. If all local validations have been successful and all vote commit messages have been received by the coordinator, the coordinator sends commit to the participants. These write the changes stored in their private workspaces to the actual data.

The disadvantage of FOCC is normally that read set and write set have to be known in advance. While this is a restriction for some application use cases, it is no problem in our application scenario.

Listing 1: Forward Oriented Optimistic Concurrency Control (FOCC) [11]; the currently validated transaction T_j is validated against all other active transactions

```
VALID: = TRUE;
FOR $T_i$ = $T_{act1}$ TO $T_{actn}$ DO
        IF WS($T_j$) $\cap$ RS($T_i$) $\neq \emptyset$ THEN
                VALID: = FALSE;
IF VALID THEN COMMIT
        ELSE RESOLVE CONFLICT;
```

This procedure is only sufficient for guaranteeing local serializability because the validation order of global transactions can be different on different nodes. The problems of distributed validation are described in detail by Schlageter [26]. One method for guaranteeing global serializability is to use timestamps and to make sure that the validation of all global transactions is performed in the same order on all participating nodes. When the local execution order on all nodes is identical, it is also the same as the global execution oder.

The problem is that this can easily fail in wireless sensor networks due to message loss and differing hop counts. Therefore, we used a short time interval (10 ms) to guarantee the arrival of the begin vote message at all participating nodes of one transaction. Then we sorted the transactions to be validated by their priorities to make sure that the transaction with the highest priority is executed first.

Transactions arriving late (with a smaller timestamp than that of the transaction already validated) are aborted. The same difficulties concerning time synchronization arise as with timestamp ordering.

An additional problem arises due to the time lag of local validation phase and write phase. Since it is not guaranteed that transactions validated successfully locally are also committed (unsure updates), the outcome of transactions reading the unsure updates is undetermined. One possibility to prevent this is the locking of unsure updates, which again leads to a blocking situation analogous to locking and timestamp ordering. Since sensor nodes are severely resource constrained devices, we again decided to abort depending transactions in this case. Our implemented version of FOCC is called validation in the following sections to improve the readability. In the next section we present our evaluation results.

III. Evaluation

In this section we compare the concurrency control protocols locking, timestamp ordering and validation in simulations performed with the network simulator Shawn [8] with respect to their usability in wireless sensor networks.

Simulation Parameter	Value
Width of area (field units)	500 FU
Height of area (field units)	500 FU
Number of nodes	100
Number of simulation runs	20
Number of iterations per run	1030
Number of transactions per run	1000
Average neighborhood size	9.84
Maximum range (field units)	100 FU

TABLE I: Constant parameters used in the simulations performed with the network simulator Shawn [8]

A. Criteria

We compare the implemented protocols with regard to the criteria commit rate, which is the number of committed transactions divided by the number of started transactions and the costs expressed in transmitted bytes per committed transaction. Our goal is to find the most efficient protocol for concurrent transaction processing in wireless sensor networks to prolong their lifetime.

B. Simulation Setup

We used the network simulator Shawn to create a simulation scenario. In this scenario, we started 1000 transactions with each protocol for comparison matters. The constant parameters used in our simulations with Shawn are shown in Table I.

To vary the loss rate, we simulated the Quasi Unit Disk Graph Model [12] with a maximum range r_{max} of 100 field units and varied the guaranteed minimal range r_{min}. The probability p that a node at a distance of d receives a message is 1 if $d < r_{min}$, 0 if $d > r_{max}$ and decreases linearly from 1 to 0 if $r_{min} \leq d \leq r_{max}$. Links were unidirectional. We also varied the number of participants of each transaction and the ratio between read-only and writing transactions. We repeated every experiment 20 times and report the averaged values with 95% confidence intervals

C. Results

The outcome of our protocol comparison is shown in Figure 1. The upper diagrams compare the commit rates at a guaranteed range of 100 field units (this means without message loss) in the left diagram and a guaranteed range of 10 field units (this means with message loss) in the right diagram. We simulated all three protocols with 10 participants and varied the percentage of write transactions from 0%, which means exclusively read-only transactions were performed, to 100%, which means every transaction performed a write operation.

No message loss and exclusively read-only transactions and hence no conflicts is the ideal scenario, which is shown in the upper left diagram. When performed with 0% write transaction, all three protocols committed over 99% of the started transactions. The missing 1% of the transactions have been aborted due to timeouts caused by a partly partitioned network in one out of our 20 simulation repetitions. When the percentage of write transactions is increased also the

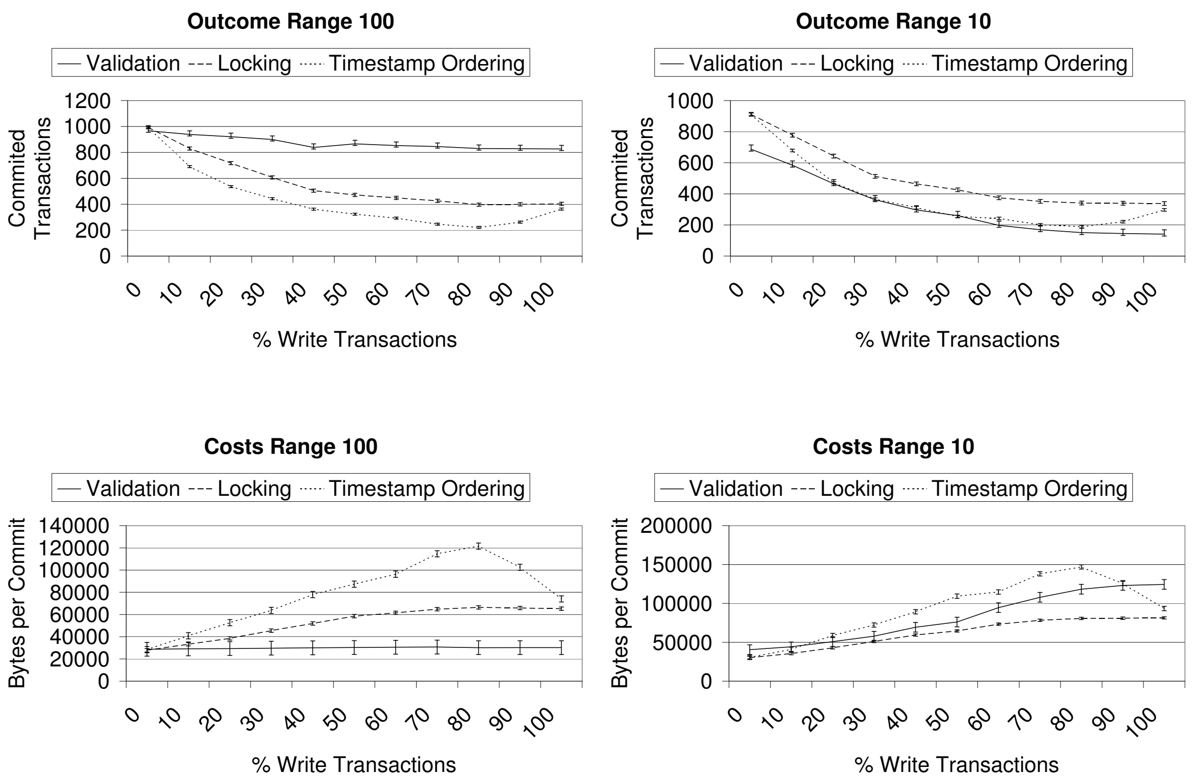

Fig. 1: Experimental results from comparing the implemented concurrency control protocols with 10 participants

probability of a conflict between two transactions increases and consequently, the commit rate decreases. It can be seen that validation is superior when simulated without message loss, while locking has the second highest commit rate and timestamp ordering performs worst. With message loss, there is a tremendous drop of the commit rate and locking becomes superior. The same can be said about the costs. While validation is most efficient without message loss, locking is most efficient with a guaranteed range of 10 field units.

Without message loss, the average commit rate of validation was 87%, locking 56% and timestamp ordering 43%. With the guaranteed ranged reduced to 10 field units, the average commit rate of validation was only 31%, locking 50% and timestamp ordering 38%. So while timestamp ordering and locking lose each about 10%, validation loses more than 50% of the commit rate. This is because the collection of begin vote message and sorting by priorities does not perform well if message loss occurs.

We used a relatively fast timing and the simulation used perfectly synchronized clocks as well as immediate message delivery, both of which are advantages for the timestamp ordering and validation. Nevertheless, locking outperformed the other protocols when simulated with message loss. Since message loss is typical for wireless sensor networks, we consider locking the best concurrency control protocol for this environment. The added advantage of locking is that no work is done in vain: a significant advantage because it further extends the lifetime of sensor nodes.

D. Experiments with Real Sensor Nodes

Simulations give only limited insight into the real world behavior of algorithms due to their simplifications of reality. Oftentimes a flat world is simulated, resource consumptions are not taken into account or one of many other potential simplifications is calculated. Our simplified simulation environment is an advantage for timestamp ordering and validation because we provided perfect time synchronization, which is not available in real wireless sensor networks. Nevertheless, the results show the advantage of locking and consequently, we did only implement locking for real sensor nodes.

We implemented two phase locking integrated in 2PC for the sensor node platform Pacemate [16], to prove the feasibility of the protocol. With a Philips LPC 2136 processor, 32 kByte RAM and 256 kByte flash ROM it has roughly the same resources as common sensor nodes, like Mica2 nodes, but offers a display and buttons for easier debugging of applications. In the experiments performed with the Pacemate sensor nodes, we gained commit rates similar to the simulation results. There are several ways to improve the commit rate. On the one hand, the acknowledgement of messages can be used and messages can be sent until they are finally received. This is a problem in the presence of node failures, since messages are often sent infinitely because node failures cannot be detected. Another way of improving the commit rate is the usage of more sophisticated commit protocols than 2PC.

We recommend the usage of our variant of 2PC called Two Phase Commit with Caching (2PCwC) [23]. In 2PCwC, participants of a transaction cache messages from other participants to be able to reply in place of them if their messages get lost. We show in experiments with 20 Pacemate sensor nodes that our protocol can increase the commit rate from 53% to 84% and also significantly lowers the costs [20]. In large sensor networks containing more than about 200 nodes, one should abstain from flooding and use routing protocols with better scalability properties like georouting [24].

IV. Conclusion

In this paper we outlined our implementations of traditional concurrency control protocols for resource constrained wireless sensor networks. We have shown that traditional locking is superior to timestamp ordering and validation when considering the commit rate and the efficiency. Beyond that, locking does not require time synchronization like timestamp ordering and validation. The additional advantage of locking is that no work is done in vain in the highly resource constrained sensor networks, which saves energy. We also implemented locking integrated in 2PC for the sensor node platform Pacemate to validate the feasibility of the protocol for resource constrained wireless sensor networks.

In our future work we are investigating how different concurrency control protocols can be used in an adaptive manner. The usage could be done adaptive to the current message loss, but also to conflicts and type of transactions. For instance, validation could be used if the message loss is really low, otherwise locking could be used.

Instead of running the concurrency protocols directly on the sensor nodes, proxies could be used to run the protocols to release this work from the resource constrained sensor nodes. Another open question is, if the consistency could be relaxed in some application domains for wireless sensor networks.

References

[1] Ian F. Akyildiz and Ismail H. Kasimoglu. Wireless sensor and actor networks: research challenges. *Ad Hoc Networks*, 2(4):351 – 367, 2004.

[2] Brahim Ayari, Abdelmajid Khelil, and Neeraj Suri. Partac: A partition-tolerant atomic commit protocol for manets. In *Proc. of The 11th International Conference on Mobile Data Management (MDM)*, 2010.

[3] Philip A. Bernstein and Nathan Goodman. Concurrency control in distributed database systems. *ACM Comput. Surv.*, 13(2):185–221, 1981.

[4] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[5] Angelo Brayner and Frank Stefan Alencar. A semantic-serializability based fully-distributed concurrency control mechanism for mobile multi-database systems. *Database and Expert Systems Applications, International Workshop on*, 0:1085–1089, 2005.

[6] Angelo Brayner and Jose Aguiar Filho. Increasing mobile transaction concurrency in dynamically configurable environments. In *Proceedings of the Third International Workshop on Mobile Distributed Computing - Volume 06*, pages 637–641, Washington, DC, USA, 2005. IEEE Computer Society.

[7] Yanlei Diao, Deepak Ganesan, Gaurav Mathur, and Prashant J. Shenoy. Rethinking data management for storage-centric sensor networks. In *CIDR*, pages 22–31. www.crdrdb.org, 2007.

[8] Sándor P. Fekete, Alexander Kröller, Stefan Fischer, and Dennis Pfisterer. Shawn: The fast, highly customizable sensor network simulator. In *Proceedings of the 4th International Conference on Networked Sensing Systems (INSS 2007)*, page 299, 2007.

[9] Jim Gray. Notes on data base operating systems. In *Operating Systems, An Advanced Course*, pages 393–481, London, UK, 1978. Springer-Verlag.

[10] Levent Guergen, Claudia Roncancio, Cyril Labbé, and Vincent Olive. Transactional issues in sensor data management. In *DMSN '06: Proceedings of the 3rd workshop on Data management for sensor networks*, pages 27–32, New York, NY, USA, 2006. ACM.

[11] Theo Haerder. Observations on optimistic concurrency control schemes. *Inf. Syst.*, 9(2):111–120, 1984.

[12] Fabian Kuhn, Roger Wattenhofer, and Aaron Zollinger. Ad hoc networks beyond unit disk graphs. *Wirel. Netw.*, 14(5):715–729, 2008.

[13] H. T. Kung and John T. Robinson. On optimistic methods for concurrency control. *ACM Trans. Database Syst.*, 6(2):213–226, 1981.

[14] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[15] Victor C. S. Lee and Kwok-Wa Lam. Optimistic concurrency control in broadcast environments: Looking forward at the server and backward at the clients. In *MDA '99: Proceedings of the First International Conference on Mobile Data Access*, pages 97–106, London, UK, 1999. Springer-Verlag.

[16] Martin Lipphardt, Horst Hellbrück, Dennis Pfisterer, Stefan Ransom, and Stefan Fischer. Practical experiences on mobile inter-body-area-networking. In *Proceedings of the Second International Conference on Body Area Networks (BodyNets'07)*, 2007.

[17] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, 2005.

[18] René Müller, Gustavo Alonso, and Donald Kossmann. Swissqm: Next generation data processing in sensor networks. In *CIDR*, pages 1–9. www.crdrdb.org, 2007.

[19] Yoav Raz. The principle of commitment ordering, or guaranteeing serializability in a heterogeneous environment of multiple autonomous resource mangers using atomic commitment. In *VLDB '92: Proceedings of the 18th International Conference on Very Large Data Bases*, pages 292–312, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

[20] Christoph Reinke. Adaptive service migration and transaction processing in wireless sensor networks. In *Proceedings of the 7th Middleware Doctoral Symposium (MDS 2010)*, pages 8–13, Bangalore, India, November 29 - December 03 2010. ACM.

[21] Christoph Reinke, Nils Hoeller, and Volker Linnemann. Adaptive atomic transaction support for service migration in wireless sensor networks. In *IEEE WOCN2010*, 2010.

[22] Christoph Reinke, Nils Hoeller, Martin Lipphardt, Jana Neumann, Sven Groppe, and Volker Linnemann. Integrating standardized transaction protocols in service oriented wireless sensor networks. In *Proceedings of the 24th ACM Symposium on Applied Computing*, pages 2202–2203, Honolulu, Hawaii, USA, March 8 - 12 2009. ACM.

[23] Christoph Reinke, Nils Hoeller, Jana Neumann, Sven Groppe, Volker Linnemann, and Simon Werner. Analysis and comparison of atomic commit protocols for adaptive usage in wireless sensor networks. In *Proceedings of The Third IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing*, 2010.

[24] Christoph Reinke, Nils Hoeller, Stefan Werner, Sven Groppe, and Volker Linnemann. Consistent service migration in wireless sensor networks. In *Proceedings of the 2011 International Conference on Wireless and Optical Communications (ICWOC 2011)*, 2011.

[25] K. Römer, P. Blum, and L. Meier:. *Handbook of Sensor Networks: Algorithms and Architectures*, chapter Time Synchronization and Calibration in Wireless Sensor Network, pages 199–237. Wiley and Sons, 2005.

[26] G. Schlageter. Problems of optimistic concurrency control in distributed database systems. In *ACM SIGMOD Record*, volume 12, pages 62–66, New York, NY, USA, 1982. ACM.

[27] Zhaowen Xing, Le Gruenwald, and Seokil Song. An optimistic concurrency control algorithm for mobile ad-hoc network databases. In *Proceedings of the Fourteenth International Database Engineering &#38; Applications Symposium*, IDEAS '10, pages 199–204, New York, NY, USA, 2010. ACM.