An Optimistic Concurrency Control Algorithm for Mobile Ad-hoc Network Databases

Zhaowen Xing
School of Computer Science
University of Oklahoma
Norman, OK 73019, USA
zhaowenxing@ou.edu

Le Gruenwald
School of Computer Science
University of Oklahoma
Norman, OK 73019, USA
ggruenwald@ou.edu

Seokil Song
Department of Computer Engineering
Chungju National University
Chungbuk, 380-702, Republic of Korea
sisong@cjnu.ac.kr

ABSTRACT
With the rapid growth of database applications, wireless networking technology and mobile computing devices, there is a demand for processing mobile transactions in Mobile Ad-hoc Network (MANET) databases, so that mobile users can access and manipulate data anytime and anywhere. However, in order to guarantee timely and correct results for multiple concurrent transactions, concurrency control (CC) techniques become critical. Due to the characteristics of MANET databases, existing CC algorithms cannot work effectively. In this paper, we propose a CC algorithm called Sequential Order with Dynamic Adjustment (SODA) for MANET databases. In the design of SODA, the major characteristics of MANET databases are taken into consideration. SODA is based on optimistic CC to offer high concurrency and avoid unbounded blocking time, utilizes the sequential order of committed transactions to improve response time, and dynamically adjusts the sequential order of committed transactions to reduce aborts. The simulation results confirm that SODA has lower abort rate than other existing techniques.

Categories and Subject Descriptors
H.2.4 [**Systems**]: Concurrency, Distributed Databases, Transaction Processing

General Terms
Algorithms, Performance, Experimentation.

Keywords
Mobile transaction processing, Optimistic concurrency control, Mobile ad-hoc networks.

1. INTRODUCTION
Mobile databases refer to database applications in which users utilize their mobile devices, such as pocket PCs, smart phones, and laptops, to manage data while they move. A mobile database system built on a Mobile Ad-hoc Network (MANET) is called a MANET database system. In this system, both clients and servers where the databases are stored and accessed by clients are mobile. As no fixed infrastructures are required, MANET databases can be deployed in a short time and end users can access and manipulate data anytime and anywhere, and thus they become an attractive solution to handle mission-critical database applications, such as disaster response and recovery systems [4, 2] and military operations like battle fields [2]. In these applications, transactions must be executed not only correctly but also within their deadlines. To guarantee this, a concurrency control (CC) technique must be a part of the system.

CC is the activity of preventing transactions from destroying the consistency of the database while allowing them to run concurrently, so that the throughput and resource utilization of database systems are improved and the waiting time of concurrent transactions is reduced [13]. However, the flexibility and convenience in a MANET introduces a number of constraints/characteristics, which impact transaction processing and are listed below. As a result of these constraints and of the fact that servers are also mobile, CC techniques for cellular mobile databases cannot be directly applied in MANET environments.

- *Mobility*: When a node roams, its network and physical location change dynamically, and at the same time, the states of transactions and accessed data items have to move along with the node.
- *Low bandwidth*: Wireless network bandwidth is much lower than its wired counterpart. For example, the widely used 802.11b wireless card has a maximum data rate of 11 Mbit/s; however, currently an affordable Gigabit Ethernet card realizes a maximum data rate of 1000 Mbit/s. Thus, within the cell of a node, inside neighbors have to share and compete for the same channel. If someone fails, it may keep sending requests until timeout. This low bandwidth can result in communication delays, a high risk of disconnections and long-lived transactions.
- *Multi-hop communication*: In a MANET, nodes can communicate with each other either directly or via other nodes that function as routers. When communication requires more hops, more power and bandwidth are consumed, and more execution time is needed to complete transactions.
- *Limited battery power*: Because of the mobility and portability, clients and servers have severe resource constraints in terms of capacity of battery. Once a node runs out of power or has insufficient power to function, communication fails, disconnections happen, execution of

IDEAS10 2010, August 16-18, Montreal, QC [Canada]
Editor: Bipin C. DESAI

transactions is prolonged, and some transactions may not be processed.

- *Frequent disconnections*: A node is disconnected when it roams freely and is out of the transmission range of all its neighbors; or it fails to compete for the channels of popular neighbors; or its battery runs out; or it runs into some failures. It is normal for a node to become disconnected in a MANET because the disconnected nodes may reconnect after some time. When disconnections happen, more transactions may be delayed or blocked, and even aborted if they are real-time and miss their deadlines.
- *Long-lived transactions*: Due to wireless communication delay, less processing power, frequent disconnections and unbounded disconnection time, transactions in MANET databases tend to be long-lived. When the execution is prolonged, the probability of conflicts with other executing transactions becomes higher and, consequently, transactions are likely blocked if a pessimistic CC method applies, or aborted if an optimistic CC method is in use.

CC research in MANET databases is still in an early stage. To the best of our knowledge, only one MANET CC algorithm has been proposed [3]; however, this algorithm not only relaxes transaction atomicity and global serializability, but also does not take into account all the identified characteristics of MANET databases. In this paper, we propose a CC algorithm called SODA, for mission-critical MANET databases that require global serializability. In the design of SODA, all listed characteristics are put into consideration. Nodes are divided into clusters where each cluster has one cluster head that is responsible for the processing of all the nodes in the cluster. SODA elects cluster heads based on their energy, mobility and workload to build a stable backbone, makes use of optimistic CC to offer high concurrency, and dynamically adjusts the sequential order of committed transactions to reduce transaction aborts.

The rest of this paper is organized as follows. Section 2 provides a brief review of recently proposed CC techniques for mobile databases. In Section 3, a model of clustered MANET databases is described. The details of SODA are presented in Section 4. The performance of SODA is analyzed in Section 5. Finally, conclusions and future research are given in Section 6.

2. RELATED WORK

Since cellular mobile networks and MANET have many similar characteristics, in this section, we review the CC techniques recently proposed for databases in both networks.

Semantic Serializability Applied to Mobility (SESAMO) [3] was proposed for MANET databases. SESAMO is based on semantic serializability, which assumes that databases are disjoint and updates on a database only depend on the values of data of the same database; therefore, transaction atomicity and global serializability can be relaxed. However, in SESAMO, global transactions still need be serialized at each site using strict 2PL; while at the same time, each site must maintain the consistency of its own local database. Look-Ahead Protocol (LAP) was proposed in [10] to maintain data consistency of broadcast data in mobile environments. In LAP, update transactions are classified into either hopeful or hopeless transactions. Hopeless transactions can not commit before their deadlines, and are aborted as earlier as possible to save system resources and reduce data locks, while hopeful transactions can continue to execute their read and write operations using the 2PL algorithm.

Multi-Version Optimistic Concurrency Control for Nested Transactions (MVOCC-NT) [11] was proposed to process mobile real-time nested transactions using multi-versions of data in mobile broadcast environments. MVOCC-NT adopts the timestamp interval with dynamic adjustment to avoid unnecessary aborts. At mobile clients, all active transactions perform backward pre-validation against transactions committed in the last broadcast cycle at the fixed server. Read-only transactions can commit locally if they pass the pre-validation, but surviving update transactions have to be transferred to the fixed server for the final validation. Choi et al. [6] proposed 2-Phase Optimistic Concurrency Control (2POCC) to process mobile transactions in wireless broadcast environments. Transaction validation is done in two phases: partial backward validation at mobile clients and final validation at the fixed server. In both phases, if a transaction T_i is serialized before transaction T_j then the writes of T_i should not overwrite the writes of T_j and the writes of T_i should not affect the reads of T_j.

Except for SESAMO, all the reviewed techniques are designed for cellular mobile databases that heavily rely on broadcast techniques and powerful servers that are static; thus, they are not suitable for MANET databases. SESAMO was proposed for MANET databases. It relaxes the atomicity and global serializability of global transactions. However, the MANET databases for mission-critical applications cannot relax the atomicity and global serializability because each database depends on each other due to the organizational structure such that semantic units cannot be defined. For example, in a disaster rescue scenario, before sending firefighters to pursue some actions, the status of their equipments has to be checked, where the firefighter database table may be stored on one mobile server, and the equipment database table may be stored on another mobile server. In a battlefield scene, before a tank fires cannon, the locations of the soldiers have to be checked to ensure their safety, where the tank database table is stored on one mobile server, and the soldiers' information is located on another mobile server; also, SESAMO does not consider all the characteristics of MANET databases described in Section 1.

3. ARCHITECTURE OF MANET DATABASES

In our MANET database architecture, depending on the communication strength, computing power and storage size, mobile nodes are classified into either clients or servers as shown in Figure 1. On clients, only the query processing modules that allow them to submit transactions and receive results are installed; while on servers, the complete database management system is installed and servers provide transaction processing services. Servers are further classified into coordinating servers or participating servers. Coordinating servers are the ones which receive global transactions, divide them into sub-transactions, forward these sub-transactions to appropriate participating servers, and maintain the ACID (Atomicity, Consistency, Isolation, and Durability) properties of global transactions. Participating servers are the ones that process sub-transactions transmitted from coordinating servers, and preserve their ACID

properties. Coordinating and participating servers are not necessarily physical units, and one server can function as both.

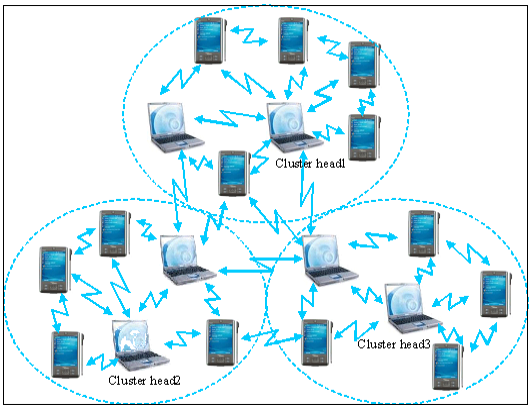

Figure 1. Architecture of a clustered MANET database

The entire database is partitioned into local databases and distributed at different servers, and there is no caching or replication technique involved for simplicity. Transactions are based on the simple flat model, which contains a set of read, write, insert, and delete operations. However, when operations are checked for conflicts, insert and delete operations are treated just as write ones. Any subset of operations of a global transaction that access the same server is submitted and executed as a single sub-transaction.

Our system is based on clustered MANETs where mobile nodes are divided into clusters and each cluster has one cluster head responsible for the processing within the cluster. We choose this architecture because of two reasons. First, many MANET applications in the literature use grouped or clustered architectures. Examples are disaster response and recovery systems using hybrid MANETs [4], and mobile telemedicine systems [14]. Second, because every node is mobile in MANET, the network topology may change rapidly and unpredictably over time. According to [5], clustered architectures are proper to keep the network topology stable as long as possible, so that the performance of routing and resource relocation protocols is not compromised.

In order to have a stable network backbone and enable our CC algorithm to run smoothly, we proposed a robust weighted clustering algorithm called PMW (Power, Mobility and Workload) [16] to form and maintain more stable clusters in MANETs. In PMW, the weight of a node is calculated by three parameters: remaining power, mobility prediction (to check if a node moves along with all its one-hop neighbors) and workload (represented by power decrease rate because nodes with heavier workload consume more energy). These three metrics are computed locally, independent of extra devices, and cover the major causes of re-clustering. In other words, PMW takes mobility, limited power and frequent disconnections into consideration.

4. SEQUENTIAL ORDER WITH DYNAMIC ADJUSTMENT

In this section, we describe our proposed CC algorithm SODA, which was originally proposed for mobile P2P databases that are centralized [15]. We first describe how SODA works without a clustered MANET database involved. Second, we discuss how SODA works in a clustered MANET database that is actually a distributed database.

4.1 Description of SODA

Inspired by the dynamic adjustment technique proposed in [9], and based on the combination of Timestamp ordering (TO), OCC, and backward validation, we propose an optimistic CC algorithm called SODA.

Assume that T_i's (i = 1, …, n) are committed transactions, and T is a validating/committing transaction. If we simply let the validation/commit order be the serialization order like traditional OCC, and if there is a read-write conflict between T and T_i, i.e., T reads a common data item d before T_i updates d, then T is aborted because two orders are different. Such aborts should be avoided if possible.

We need a dynamic order among committed transactions other than the validation order; that is, in SODA, a Sequential Order (SO) of committed transactions is maintained as {T_1, T_2, …, T_i, ..., T_n} (also called a history list, which is ordered from left to right) and can be dynamically adjusted. The dynamic adjustment consists of simple and complex cases. In the simple case, the validating transaction T can be directly inserted into the maintained sequential order without adjustment, and the final sequential order will be: {T_1, T_2, …, ***low*, … T, *up***, ..., T_n}, such that T must-be-serialized-after *low* but before *up*. On the other hand, in the complex case, the sequential order must be adjusted before the insertion of T.

Due to space limitation, the detailed description, algorithm pseudo code and example for the complex case can be found in Section 4 of [15].

Example 1 (for the simple case): Let {T_1, T_2, T_3} be the sequential order of committed transactions, and T be a validating transaction at a server. The read sets, write sets and the timestamps are shown in Table 1.

Table 1. Transaction information used in Example 1

	T_1	T_2	T_3	T
Read Set (RS)	{x}	{y}	{x, y}	{x}
Write Set (WS)	{z}	{x}	∅	{z}
Read Timestamp of Data d (TS(d))	5	15	25, 30	18
Timestamp of Write Set (WS_TS)	10	20		+∞

Since $WS(T_1) \cap WS(T) \neq \emptyset$ and $WS\_TS(T) > WS\_TS(T_1)$, T must-be-serialized-after T_1 and *low* = T_1. Since $WS(T_2) \cap RS(T) \neq \emptyset$ and $WS\_TS(T_2) > T \rightarrow TS(x)$, T must-be-serialized-before T_2 and *up* = T_2. Since SO(*low*) < SO(*up*), this is the simple case and T passes the validation test. T is inserted immediately before T_2, and the final sequential order is {T_1, T, T_2, T_3} as shown in Figure 2, where the arrow (→) in the graph indicates the serialization order between two transactions such as $T_1 \rightarrow T_2$ means that T_1 must-be-serialized-before T_2.

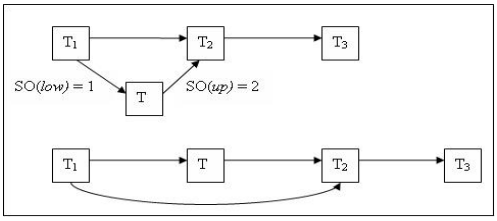

Figure 2. Validating transaction T in Example 1

4.2 How SODA Works in a Clustered MANET Database

In order to make SODA work effectively in clustered MANET databases, the coordinating server functionality is combined in the cluster head because a cluster head is elected by our PMW algorithm [15] and is the nearest server with the highest power in the neighborhood, so that this saves time, limited power and bandwidth that clients must spend to identify suitable servers to send their transaction to. Therefore, only two major functionalities are required: the participating server functionality and cluster head functionality as shown in Figure 3. Note that one server can have both the functionalities at the same time.

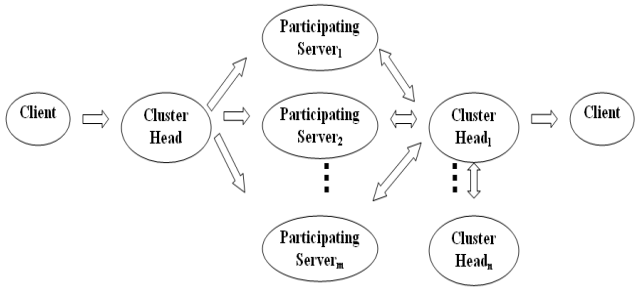

Figure 3. Transaction flow in a clustered MANET

4.2.1 The participating server functionality:

- It maintains the sequential order of committed sub-transactions. It receives and processes sub-transactions.
- When it receives the request about the status of sub-transactions, it runs SODA locally based on the local sequential order of committed transactions.
- It sends the final result to the requesting cluster head. If a sub-transaction passes the validation, the timestamp of the read set is also sent to the cluster head.
- If a sub-transaction commits, it rebuilds the local latest sequential order of committed transactions by running the algorithm update_sequential_order() [15], and adds the read set, write set and timestamps of both sets.
- Once a sub-transaction commits, it tries to remove committed transactions (or called outgoing nodes in the local serialization graph), which are not serialized after any active/committed transaction, from the local sequential order of committed transactions.

4.2.2 The cluster head functionality:

- It maintains the sequential order of committed global transactions.
- It receives global transactions from clients and divides them into sub-transactions, which are sent to appropriate participating servers based on the global schema.
- It runs 2-Phase Commit to request the status of sub-transactions, and at the same time, requests timestamps of the read set.
- Once it receives all successful messages of a global transaction and can be in the critical section after communicating with other cluster heads, it begins to validate this transaction globally using SODA. After validation, it sends the validation results to each participating server, and sends the final results to clients.
- When a global transaction commits, it updates its sequential order by running the algorithm update_sequential_order() [15], and adds the read set, write set and the timestamp of both sets.
- Once a global transaction commits, it tries to remove committed transactions (or outgoing nodes in the global serialization graph), which are not serialized after any active/committed transaction, from the sequential order of committed global transactions.
- It periodically checks its power level. If the level is below a predefined threshold, it resigns its cluster head status and elects a new cluster head in the neighborhood.

5. CORRECTNESS PROOF AND PERFORMANCE EVALUATION

In this section, we first show the correctness proof and the time complexity of SODA, and then evaluate its performance by means of simulation.

5.1 Correctness Proof and Time Complexity

Theorem 1: If S is a schedule produced by SODA, then S is serializable. The proof can be found in [15].

Theorem 2: The time complexity of SODA is $O(p*n^2 + n)$, where n is the number of committed transactions in the sequential order, and p is the probability of a committing transaction conflicting with both transactions *low* and *up* and $SO(low) \geq SO(up)$. The proof can be found in [15].

As most of transactions in MANET are read-only, the value of p will be very small, and thus, p^2 will be even smaller. Therefore, we can safely claim that SODA mostly runs in the linear time. In contrast, the complexity of a serialization graph testing algorithm is always $O(n^2)$ [9].

5.2 Performance Evaluation

Simulation experiments were conducted to compare the performance of our proposed SODA with those of SESAMO [3] and the most widely used CC protocol - Strict 2-Phase Locking (S2PL). As we discussed earlier, SESAMO is the only CC proposed for MANET databases, but it relaxes transaction atomicity and global serializability due to its database assumptions. Global serializability is guaranteed by S2PL when S2PL is combined with 2-Phase Commit (2PC) [1].

Our simulation model consists of a transaction generator, a real-time scheduler that schedules transactions using early deadline first, participating servers, coordinating servers or cluster heads for SODA only, and a deadlock manager for SESAMO and S2PL.

In the SODA model, as shown in Figure 3, a transaction T issued by a client is distributed to this client's cluster head; the cluster head divides a transaction into several sub-transactions, and transmits them to the appropriate participating servers. Each participating server processes the sub-transactions locally, and sends the results back to the cluster head. The cluster head runs the 2PC, gathers all results from the participating servers and validates T globally.

The simulation models for SESAMO and S2PL are similar to that for SODA except for two points: 1) SODA is applied locally and globally to validate transactions, while in SESAMO, strict 2PL is applied globally [3] and locally [8], and in S2PL, strict 2PL is run only locally; and 2) SESAMO and S2PL use coordinating servers instead of cluster heads.

Our simulation is implemented using the AweSim simulation language [12]. The simulation experiment consists of 20 servers and 40 clients [7]. Totally 1000 transactions are generated for each run, and the results are calculated as averages of multiple independent runs. Due to space limitation, only abort rate defined in Equation (1) is presented in this paper. Since in mission-critical applications, transactions should be executed not only correctly but also within their deadlines, where deadline = creation time + (estimated execution time + estimated disconnection time)*slack factor. In other words, we use real-time transactions to evaluate the performance. Therefore, in our simulation, a transaction will be aborted if either it misses its deadline or the system could not complete it successfully (e.g. when it is aborted by the CC technique).

$$Abort\ rate = \frac{Total\ number\ of\ aborted\ transactions}{Total\ number\ of\ generated\ transactions} * 100\% \quad (1)$$

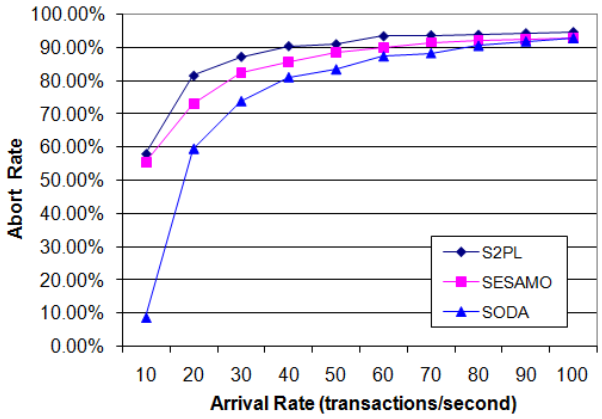

Figure 4. Abort rate vs. arrival rate

Figure 4 shows that the abort rates of SODA, SESAMO and S2PL increase when the transaction arrival rate increases. The abort rate of SODA is much lower than those of SESAMO and S2PL when the arrival rate is low. This is mainly because SODA is optimistic and non-blocking. SESAMO's abort rate is slightly lower than S2PL's. Although SESAMO does not enforce global serializability, strict 2PL running both locally and globally still blocks many conflicting transactions. S2PL runs strict 2PL locally only, but it enforces global serializability using 2PC, so that all locks of sub-transactions are held until global transactions commit, which increases the waiting time of conflicting transactions in S2PL. When the arrival rate is getting larger, the abort rates of these three algorithms are almost the same. Also it is easy to see that the abort rate of SODA is getting worse quickly when the contention of transactions increases.

Figure 5 shows that the abort rates of the three algorithms increase as the disconnection probability increases. Disconnection probability 0.5 means that 50% of communications become disconnected when a mobile node tries to communicate with other nodes. As we mentioned in Section 1, one of the major characteristics of MANET is frequent disconnections due to the mobility and energy limitation of nodes and unreliable wireless communication between nodes.

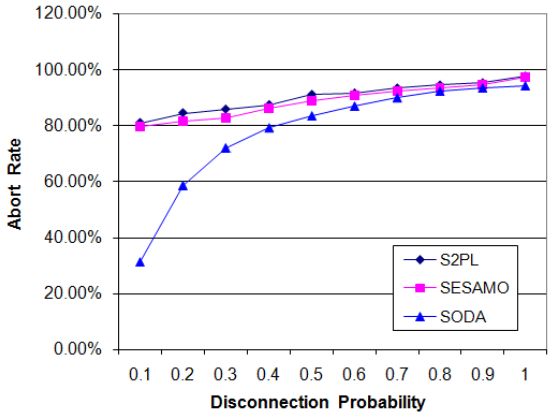

Figure 5. Abort rate vs. disconnection probability

As shown in Figure 5, the abort rate of SODA is at least 5% lower than those of SESMO and S2PL when disconnection probability is <= 0.5 because SODA uses stable cluster heads as coordinating servers, is optimistic so that it does not block transactions, and utilizes the dynamic adjustment of sequential order of committed transactions to reduce aborts. The abort rate of SESAMO is still slightly lower than S2PL's due to the same reason discussed above, and they are almost the same when the disconnection probability is 1. Since SESAMO has at least 79% abort rate, this shows that SESAMO does not address frequent disconnections.

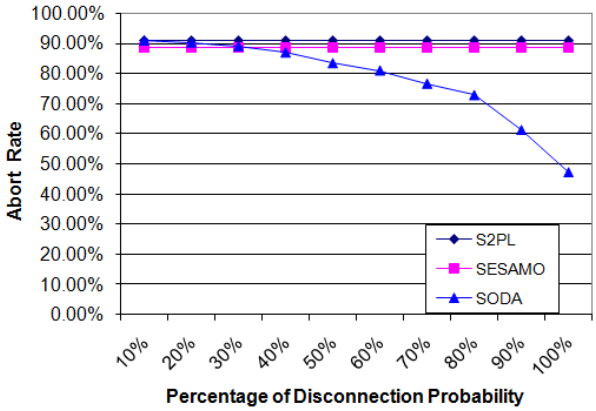

Figure 6. Abort rate vs. percentage of disconnection probability

Since in a clustered MANET, cluster heads are more stable than non-cluster head nodes, cluster heads may have less disconnection probability than non-cluster head nodes. In order to study how

less disconnection probability that cluster heads can have comparing to non-cluster head nodes, we test the percentage of disconnection probability that cluster heads can have from 10% to 100%. For example, if the default disconnection probability is 0.5 and cluster heads can have 50% of disconnection probability, then cluster heads will have only 0.25 chances to get disconnected. Figure 6 shows that the abort rate of SODA decreases as the percentage of disconnection probability that cluster heads can have increases. This is mainly because cluster heads become more stable when they have fewer disconnections. In other words, fewer transactions are aborted if more stable nodes are elected as the cluster heads. However, the abort rates of SESAMO and S2PL do not change because their designs do not involve cluster heads.

6. CONCLUSIONS

Many excellent CC techniques have been proposed for cellular mobile databases; but little research has been done for CC in MANET databases. In this paper, we proposed a CC algorithm called SODA for MANET databases. In SODA, all the identified characteristics of MANET databases are taken into account. SODA is based on optimistic CC to offer high concurrency and dynamically adjusts the sequential order of transactions to reduce transaction abort rate. The simulation experiments showed that the transaction abort rate incurred by SODA is lower than those incurred by the existing techniques. As future work, we will conduct simulation experiments to study other performance metrics, including the energy consumption by clients and servers as well as the balance in energy consumption among nodes. We will also incorporate data replication into our model to improve the data access time and availability.

7. ACKNOWLEDGMENTS

This material is based upon work supported by (while serving at) the National Science Foundation (NSF) and the NSF Grant No. IIS-0312746. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

8. REFERENCES

[1] M. Abdouli, B. Sadeg, L. Amanton and A. M. Alimi. A System Supporting Nested Transactions in DRTDBSs. In *Proceedings of the 1st International High Performance Computing and Communications*, pp. 888-897, September 2005.

[2] S. P. Alampalayam, and S. Srinivasan. Intrusion Recovery Framework for Tactical Mobile Ad hoc Networks. *The International Journal of Computer Science and Network Security*, Vol.9, No.9, pp. 1-10, September 2009.

[3] A. Brayner, and F. S. Alencar. A Semantic-serializability Based Fully-Distributed Concurrency Control Mechanism for Mobile Multi-database Systems. In *Proceedings of the 16th International Workshop on Database and Expert Systems Applications*, pp. 1085-1089, August 2005.

[4] T. Catarci, M. de Leoni, A. Marrella, M. Mecella, B. Salvatore, G. Vetere, S. Dustdar, L. Juszczyk, A. Manzoor, and H. Truong. Pervasive Software Environments for Supporting Disaster Responses. *IEEE Internet Computing*, Vol. 12, No. 1, pp. 26-37, January 2008.

[5] M. Chatterjee, S. Das, and D. Turgut. WCA: A Weighted Clustering Algorithm for Mobile Ad Hoc Networks. *Cluster Computing*, Vol. 5, No. 2, pp. 193-204, April 2002.

[6] M. Choi, W. Park and Y. Kim. Two-phase Mobile Transaction Validation in Wireless Broadcast Environments. In *Proceedings of the 3rd International Conference on Ubiquitous Information Management and Communication*, pp. 32-38, January 2009.

[7] L. Gruenwald, S. M. Banik, and C. N. Lau. Managing real-time database transactions in mobile ad-hoc networks. *Distributed and Parallel Databases*, Vol. 22, No. 1, pp. 27-54, August 2007.

[8] M. Holanda, A. Brayner, and S. Fialho. Introducing self-adaptability into transaction processing. In *Proceedings of the 2008 ACM symposium on Applied Computing*, pp. 992-997, March 2008.

[9] S. Hwang. On Optimistic Methods for Mobile Transactions. *Journal of Information Science and Engineering*, Vol. 16, No. 4, pp. 535-554, July 2000.

[10] K. Lam, C. S. Wong and W. Leung. Using Look-ahead Protocol for Mobile Data Broadcast. In *Proceedings of the 3rd International Conference on Information Technology and Applications*, pp. 342-345, July 2005.

[11] X. Lei, Y. Zhao, S. Chen, and X. Yuan. Scheduling Real-Time Nested Transactions in Mobile Broadcast Environments. In *Proceedings of the 9th International Conference for Young Computer Scientists*, pp. 1053-1058, November 2008.

[12] A. Pritsker, and J. O'Reilly. *Simulation with Visual SLAM and AweSim*, 2nd edition. New York: John Wiley & Sons, 1999.

[13] A. Silberschatz, H. F. Korth, and S. Sudarshan. *Database Systems Concepts*, McGraw-Hill College, 2005.

[14] D. V. Viswacheda, M. S. Arifianto, and L. Barukang. Architectural Infrastructural Issues of Mobile Ad hoc Network Communications for Mobile Telemedicine System. In *Proceedings of 4th International Conference on Sciences of Electronic, Technologies of Information and Telecommunications*, March 2007.

[15] Z. Xing, L. Gruenwald, and K. K. Phang. SODA: an Algorithm to Guarantee Correctness of Concurrent Transaction Execution in Mobile P2P Databases. In *Proceedings of the 19th International Conference on Database and Expert Systems Application Workshop*, pp. 337-341, September 2008.

[16] Z. Xing, L. Gruenwald, and K. K. Phang. A Robust Clustering Algorithm for Mobile Ad-hoc Networks. To appear in the book "*Handbook of Research on Next Generation Networks and Ubiquitous Computing*", Editor Samuel Pierre, IGI Global, 2010.