An Energy-efficient Concurrency Control Algorithm for Mobile Ad-hoc Network Databases

Zhaowen Xing and Le Gruenwald

School of Computer Science
University of Oklahoma
Norman, OK 73019, USA
{zhaowenxing, ggruenwald}@ou.edu

Abstract. MANET does not require any fixed infrastructure, thus it fits well in disaster rescue and military operations. However, when a node has no or insufficient energy to function, communication may fail, disconnections may happen, and transactions may be aborted if they are time-critical and miss their deadlines. Energy-efficient transaction management becomes an important issue in MANET database applications. In this paper, we propose an energy-efficient concurrency control (CC) algorithm for MANET databases in a clustered network architecture where nodes are divided into clusters, each of which has a node, called cluster head, responsible for the processing of all nodes in the cluster. In our algorithm, in order to conserve energy and balance the energy consumption among servers, we elect cluster heads to work as coordinating servers, and propose an optimistic CC algorithm to offer high concurrency and avoid wasting limited system resources. The simulation results confirm that our technique performs better than existing techniques.

Keywords: Mobile ad-hoc network, clustering, transaction management, concurrency control.

1 Introduction

A Mobile Ad-hoc Network (MANET) is a collection of mobile, wireless and battery-powered nodes, and every node can roam freely. A mobile database system built on a MANET is called a MANET database system. In this system, both clients and servers are mobile, wireless and battery-powered, and the databases are stored at servers and accessed by clients. As no fixed infrastructure is required, MANET databases can be deployed in a short time and mobile users can access and manipulate data anytime and anywhere, and they become an attractive solution for handling mission-critical database applications, such as disaster response and recovery systems [1, 2, 3] and military operations like battlefields [1]. 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 the database system are improved and the waiting time of concurrent transactions is reduced [4]. However, because of their mobility and portability, mobile nodes have severe resource constraints in terms of battery capacity, memory size and CPU speed. As the battery capacity is limited, it compromises the ability of each mobile node to support services and applications [5]. Also battery technology is not developed as rapidly as mobile devices and wireless technologies, so that the limited battery lifetime is always a bottleneck for the development of improved mobile devices [6]. Therefore, a suitable CC algorithm for MANET databases should be energy-efficient.

Although there are CC algorithms proposed for cellular mobile network databases [7, 8, 9], to the best of our knowledge, only one CC algorithm has been proposed for MANET databases [10]; however, this algorithm not only relaxes transaction atomicity and global serializability, but also does not take energy efficiency into account. To fill this gap, in this paper we propose an optimistic CC algorithm, called Sequential Order with Dynamic Adjustment (SODA), which takes mobility, real-time constraints and energy efficiency into consideration, to handle mission-critical databases in a clustered MANET architecture. Our objective is to minimize energy consumption of each mobile node, and balance energy consumption among servers, so that mobile nodes with low energy do not run out of energy quickly, and thus, the number of disconnections and transaction aborts due to low energy or energy exhaustion can be reduced. The rest of the paper is organized as follows. Section 2 reviews some of recent CC algorithms for mobile databases. Section 3 describes the proposed MANET architecture. Section 4 presents our CC algorithm, SODA. Section 5 provides the simulation results. Finally Section 6 concludes the paper with future research.

2 Related Work

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

Look-Ahead Protocol (LAP) was proposed in [8] 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 early as possible to save system resources and reduce data locks, while hopeful transactions can continue to execute their read and write operations using the two-phase locking (2PL) algorithm.

Multi-Version Optimistic Concurrency Control for Nested Transactions (MVOCC-NT) [9] 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. [7] 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 affect the reads of T_j.

All the CC techniques reviewed above were designed for cellular mobile databases, which heavily rely on broadcast techniques to save mobile nodes' energy and on static servers that have no energy limitation; thus, they are not suitable for MANET databases.

Semantic Serializability Applied to Mobility (SESAMO) [10] was proposed for MANET databases. SESAMO is based on semantic serializability, which requires that not only databases on mobile nodes be disjoint but also updates on a database depend only on the values of the data in 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 two-phase locking (S2PL), while at the same time each site must maintain the consistency of its own local database. SESAMO does not take energy efficiency into account. In addition, in MANET databases for mission-critical applications, the assumption for semantic serializability does not hold because each database depends on each other due to the organizational structure of the applications. For example, in a disaster rescue scenario, before sending firefighters out to pursue some actions, the status of their equipment has to be checked, where the firefighter database may be stored on one mobile server, and the equipment database may be stored on another mobile server.

3 Proposed Architecture

In this section, we introduce our clustered MANET architecture which is built by applying our robust weighted clustering algorithm, called MEW (Mobility, Energy, and Workload) [11]. MEW takes mobile nodes' mobility, energy and workload into consideration when grouping mobile nodes into clusters a MANET. In this architecture, mobile nodes are divided into clusters, each of which has one cluster head working as the coordinating server responsible for the transaction processing of the mobile nodes, called cluster members, within the cluster. Cluster heads can communicate with each other through some mobile nodes that work as gateways. Similarly, mobile nodes from different clusters can also communicate with each other, but they have to go through their cluster heads to get the destination addresses first. Also, nodes are put into the same cluster based on their application semantics.

We choose this architecture for three reasons. First, in many MANET applications, such as disaster response and recovery systems [2, 3] and military operations [12], users are logically grouped. Second, because every node is mobile in a MANET, the network topology may change rapidly and unpredictably over time. According to [13], 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. Third, in order to accommodate our optimistic CC algorithm SODA to guarantee global serializability, the information of committed

global transactions is maintained by the cluster head with the highest remaining energy.

Fig. 1 shows an example of a clustered MANET database architecture with three clusters, each of which is represented by a large solid circle with mobile clients and servers shown as PDA/iphone and laptop icons, respectively. The arrows between the devices show the communication between them. In the rest of this section, we describe the functionality of mobile nodes (Section 3.1), the MEW algorithm (Section 3.2), the cluster formation (Section 3.3), and the cluster maintenance (Section 3.4).

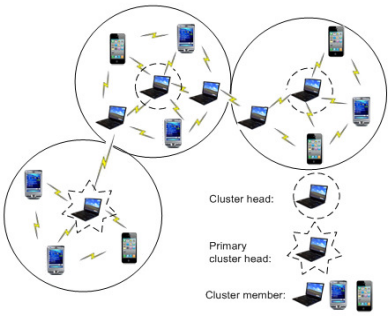

Fig. 1. Architecture of a clustered MANET database

3.1 Mobile Node Functionality

Depending on the communication strength, computing power and storage size, mobile nodes are classified into clients and servers. 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 systems are installed to provide transaction processing services. Servers are further classified into coordinating servers and participating servers. Coordinating servers are the ones which receive global transactions from clients, divide them into sub-transactions, forward these sub-transactions to appropriate participating servers, and maintain the ACID (Atomicity, Consistency, Isolation, and Durability) [4] properties of global transactions. Participating servers are the ones that process sub-transactions transmitted from coordinating servers, and preserve their ACID properties.

The entire database is partitioned into local databases and distributed to 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. Any subset of operations of a global transaction that access the same server is submitted and executed as a single sub-transaction.

With respect to the clustered MANET architecture shown in Fig. 1, a mobile node is either a cluster head when it is a coordinating server or is a cluster member when it is either a participating server or a client. In order to guarantee global serializability and reduce communication overhead, among cluster heads the one with highest remaining energy is further elected as the primary cluster head, which maintains the information of committed global transactions and validates transactions globally.

3.2 The Basis of Our Clustering Algorithm - MEW

Being inspired by the mobility based clustering algorithm, MOBIC [14], and the weighted clustering algorithm, WCA [13], and considering a new system parameter "Energy Decreasing Rate" (*EDR*), we propose a weighted clustering algorithm, called MEW (Mobility, Energy, and Workload), to build a stable backbone in MANETs. The objective of MEW is to form and maintain stable clusters in MANETs by electing nodes with the highest weights as cluster heads, where the weight of a node is calculated as a combination of its mobility, energy and workload.

To capture the mobility of nodes, we do not consider their absolute roaming speed. This is because it is easy to calculate the speed's quantity, but it is hard to predict the direction of movement. Without the direction, the speed's quantity alone is not appropriate to justify whether a node is a good candidate for cluster head or not. For instance, two nodes that have small speeds move in the opposite directions. As time goes, they will be out of each other's transmission range and get disconnected from each other. Also the utilization of GPS is opted out because GPS consumes the limited battery energy of mobile nodes.

Instead, two mobility metrics, Relative Mobility between two nodes i and j (RM_{ij}) [14] and Mobility Prediction for node j (MP_j), are introduced to monitor the mobility of nodes and applied to determine whether a node is suitable to be a cluster head. RM_{ij} measures whether node i and node j move relatively together; MP_j measures whether all 1-hop neighbors of node j move relatively together along with node j. Below we explain how each of the two metrics is calculated.

For each node j ($1 \le j \le N$ for N nodes in the network), after receiving two successive HELLO messages from every 1- hop neighbor i ($1 \le i \le n$ if there are n neighbors), the RM_{ij} is calculated by Equation (1). RSS_{ij1} and RSS_{ij2} are the received signal strengths (*RSS*) that are read from the *RSS* indicator when the first and second HELLO messages from the same neighbor i are received by node j, respectively. Based on the value of RM_{ij}, we can say that if RM_{ij} is equal to 1, then the node j and its neighbor i either do not move at all or move with the same speed in the same direction; if RM_{ij} is less than 1, then they move close to each other; otherwise, they move away from each other.

$$RM_{ij} = \frac{RSS_{ij1}}{RSS_{ij2}} \tag{1}$$

For each node j, in order to take into account the mobility of all n 1-hop neighbors and have an integral value to represent them, MP_j is calculated as the standard deviation of $RM_{1j}, RM_{2j}, \ldots, RM_{nj}$ shown in Equation (2). However, for the stability of elected clusters, we prefer RM_{ij} to be equal to or less than 1 because we want cluster heads not to move away from their members. Thus, in the MP_j calculation, the mean of RM_{ij} ($1 \le i \le n$) is 1 instead of the actual mean. A node j with a lower MP_j means that it stays closer to its neighbors, thus, it is a better candidate for the cluster head among its neighbors.

$$MP_j = \sqrt{\frac{\sum_{i=1}^{n} (RM_{ij} - \overline{RM_{ij}})^2}{n}}, \text{ where } \overline{RM_{ij}} = 1 \tag{2}$$

When dealing with the limited battery energy, we consider not only the Remaining Energy (*RE*) of each node but also its Energy Decrease Rate (*EDR*) as the workload because nodes with a heavier workload consume more energy, so that we can balance the energy usage and prevent cluster heads from running out of energy quickly. In other words, for each node *j*, EDR_j is considered because RE_j represents only the current state of the energy level and the node's energy will run out soon if it normally has a heavy workload. The EDR_j at time interval $[t_1, t_2]$ is calculated by using Equation (3), where RE_{j1} and RE_{j2} are the remaining energy of node *j* at time t_1 and t_2, respectively.

$$EDR_j = \frac{RE_{j_1} - RE_{j_2}}{t_2 - t_1} \tag{3}$$

A node with a lower *EDR* indicates that it was not busy at least during the interval $[t_1, t_2]$. However, when a node has a busy work history, it most likely will be busy in the future as well. Since the larger the time interval is, the more accurate the *EDR* is in indicating a node's workload history, during the initial election, each node saves a copy of its initial remaining energy and initial time as RE_{j1} and t_1, so that a more accurate *EDR* can be calculated in the future cluster head reelection.

Based on the above analysis about mobility, energy and workload, it is obvious that a node *j* is the best candidate for a cluster head among all its neighbors if its RE_j is the highest, its MP_j is the lowest and its EDR_j is the lowest. In other words, a node with the highest weight is the best candidate for a cluster head when we combine these three metrics together as the weight, which is calculated in Equation (4). Since these metrics have different units, we apply the inversed exponential function to normalize MP_j and EDR_j and bound their values between 0 and 1. RE_j is left out because the value of the remaining energy is at most 100%.

$$W_j = f_1 * e^{-MP_j} + f_2 * RE_j + f_3 * e^{-EDR_j} \tag{4}$$

In Equation (4), $RE_j = RE_{j2}$, the weighting factors f_1, f_2 and f_3 are set according to different application scenarios, and $f_1 + f_2 + f_3 = 1$. When we let $f_2 = f_3 = 0$, that is, we take away the effect of energy and workload, our algorithm turns into a mobility-only-based approach just like MOBIC [14].

3.3 Cluster Formation

To form clusters, each node first broadcasts HELLO messages, collects 1-hop neighbors' information, and computes its weight based on mobility, energy and workload using Equations (1), (2), (3) and (4) defined above. After broadcasting their own weights and receiving all 1-hop neighbors' weights, nodes with the highest weights declare themselves as cluster heads, and 1-hop neighbors of cluster heads join them as cluster members. The details of the cluster formation and different types of messages used in cluster formation are presented in [11]. Note that because clients have less communication strength, less computing power and smaller storage size than servers, clients cannot be elected as cluster heads and cannot work as coordinating servers.

3.4 Cluster Maintenance

Because every node can roam and has limited battery power in a MANET, cluster heads can resign due to low remaining energy, the links between cluster members and cluster heads can be broken, and the links between two cluster heads can be generated [15]. Consequently, clusters need be re-clustered. In other words, leaving clusters, joining clusters, merging clusters, and re-electing cluster heads are normal re-clustering operations in a clustered MANET. However, these operations should be performed only on demand to reduce the overhead of computation and communication, and to provide consistent quality of service.

In order to maintain connections with neighbors, detect the link breaks and new link establishments, each node needs periodically broadcast HELLO messages. Being a cluster head, it has to periodically monitor (after a global transaction commits) its remaining energy level so that it will resign from the cluster head status when the remaining energy drops below a predefined Low Energy Threshold (*LET*). Relying on these two periodic operations, cluster maintenance can be done by recovery from a link break between a member and its cluster head, recovery from a link establishment when two cluster heads become 1-hop neighbors, or recovery from a link break when a cluster head resigns because its current remaining energy of a cluster head is less than *LET*. The details of these recovery tasks are discussed in [11].

4 Proposed Concurrency Control Algorithm: SODA

In this section, we describe our proposed CC algorithm, called Sequential Order with Dynamic Adjustment (SODA). We first show how SODA works in a centralized database as originally presented in [17]. We then discuss how SODA works in a clustered MANET database.

In [17], SODA is proposed for mobile P2P databases, in which each peer carries its own local database, is fully autonomous and shares information in on-the-fly fashion. Therefore, although global transactions (or called remote queries) do exist, it is unnecessary to maintain the global serializability among peers, which is required in traditional distributed databases and MANET databases that we focus on in this paper. However, in mobile P2P databases every peer still needs to guarantee the correctness of transactions that it processes locally because it may collect or update its own data, reply to requests and update replica simultaneously.

4.1 How SODA Works in a Centralized Database

Inspired by the dynamic adjustment technique proposed in [18], and based on the combination of Timestamp Ordering (TO), Optimistic Concurrency Control (OCC), and backward validation, we propose an optimistic CC algorithm called SODA. In SODA, a list of committed transactions is maintained to validate committing transactions. During the validation, the list can be dynamically adjusted to avoid unnecessary aborts. After the committing transaction commits, the list is updated and trimmed.

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 in 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.

To avoid such aborts, in SODA, a dynamic order instead of validation order among committed transactions is used. In SODA, a Sequential Order (SO) of committed transactions is maintained as $\{T_1, T_2, \ldots, T_i, ..., T_n\}$ (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 commit if it can be directly inserted into the maintained sequential order without adjustment, and the final sequential order will be $\{T_1, T_2, \ldots,$ ***low*****, … T,** ***up***, ..., $T_n\}$, such that T must-be-serialized-after the transaction *low* but before the transaction *up*. On the other hand, in the complex case, the sequential order must be adjusted before the insertion of T. After T passes the validation and commits, the maintained SO is updated with T's information. In addition, old committed transactions are removed from the maintained SO to reduce the overhead and save the limited storage.

To prove the correctness (or completeness) of SODA, we must show that any schedule produced by SODA is serializable. To fulfill this goal, we proved that the new serialization graph is still acyclic after the addition of any newly committed transaction that has passed our validation test. Further details can be found in [17].

4.2 How SODA Works in a Clustered MANET Database

In order to make SODA work effectively in a clustered MANET database, the coordinating server functionality is combined with the cluster head's functionality because a cluster head is elected by our MEW algorithm as described in Section 3 and is the nearest server with the highest energy in clients' neighborhood. This would enable clients to save time, limited battery energy and bandwidth that they must spend on identifying suitable servers to which they send their transactions. Therefore, only three major functionalities are required: the primary cluster head functionality, cluster head functionality, and participating server functionality as shown in Fig. 2. Note that one server can have all the three functionalities at the same time.

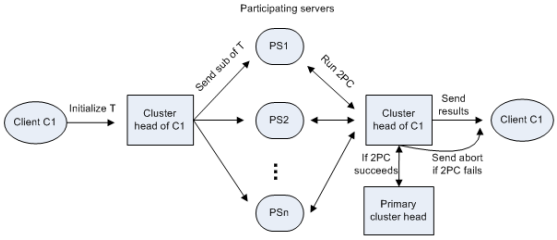

Fig. 2. Transaction flow in a clustered MANET database

4.2.1 Transaction execution model

As shown in Fig. 2, a transaction T issued by a client is distributed to its cluster head; the cluster head divides T into sub-transactions and transmits them to the appropriate participating servers according to the global database schema. Each participating server processes the sub-transactions locally and sends the results back to the cluster head. The cluster head runs the 2-Phase Commit (2PC), and gathers all results from the participating servers. Note that we adopt 2PC here due to its simplicity as our research goal is to develop a concurrency control algorithm, not a commit algorithm; however, we do plan to include a more suitable commit protocol for MANET databases in our future work. If running 2PC successfully, the cluster head sends T to the primary cluster head to validate T globally based on the SO of committed global transactions; otherwise, the cluster head sends an abort message directly to the client. After receiving the global validation result, the cluster head sends the final results to the client.

4.2.2 The primary cluster head functionality

The primary cluster head has the following functionalities:

- It maintains the sequential order (SO) of committed global transactions.
- It receives global transaction validation requests from non-primary cluster heads.
- It validates global transactions using SODA. After validation, it sends the validation results to the non-primary cluster head.
- It updates the SO after a global transaction commits and adds this global transaction's read set, write set and the timestamp of both sets to the data structure of the maintained SO.
- It removes the old committed transactions that are not serialized after any active/committed global transaction from the maintained SO after a global transaction commits.
- It periodically checks (after a global transaction commits) its remaining energy level. If its level is below a predefined threshold *LET* and another cluster head's remaining level is above the threshold, it resigns its cluster head status and elects a new primary cluster head that has the highest remaining energy from all cluster heads. It then transfers the information of all the transactions it stores to the new primary cluster head. Note that since the primary cluster head is also a non-primary one, if the primary one resigns, the non-primary one also resigns if there is a candidate in the neighborhood.

4.2.3 The non-primary cluster head functionality

A non-primary cluster head has the following functionalities:

- It receives a global transaction from a client, divides them into sub-transactions, and sends the sub-transactions to appropriate participating servers.

- It runs 2PC to request the status of the sub-transactions and requests the timestamps of the global transaction's read set.
- It propagates the global transaction to the primary cluster head after it receives all successful messages of the sub-transactions. After receiving the validation result, it sends the final results to the client.
- It periodically checks (after a global transaction commits) its remaining energy level. If the level is below a predefined threshold and there is a candidate for cluster head in the neighborhood, it resigns its cluster head status and elects a new cluster head in the neighborhood. It then transfers the information of all the transactions it stores to the new cluster head. Note that if the old cluster head is also the primary cluster head, then the new cluster head can be the new primary cluster head as well if this new one has the highest remaining energy among all cluster heads.

4.2.4 The participating server functionality

A participating server has the following functionalities:

- It receives and processes sub-transactions, and maintains the SO of committed sub-transactions.
- It runs SODA locally based on the local SO of committed sub-transactions when it receives the request about the status of the sub-transactions.
- It sends the final status of the sub-transactions to the requesting cluster head. It also sends the timestamps of the read sets of the sub-transactions to the cluster head if the sub-transactions pass the validation.
- It updates the local SO of committed sub-transactions if a sub-transaction commits and adds this sub-transaction's read set, write set and timestamps of both sets to the data structure of the maintained SO. It removes the old committed sub-transactions that are not serialized after any active/committed sub-transaction from the maintained SO after a sub-transaction commits.

5 Performance Evaluation

The simulation experiments are conducted to compare the performance of our proposed SODA with that of SESAMO [10] as unlike other existing CC for cellular mobile databases, SESAMO was specifically designed for MANET databases.

Our simulation models consist 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. The simulation model of SODA is the same as the transaction execution model discussed in Section 4.2.1. The simulation model for SESAMO is similar to the one of SODA except for a couple of points. First, SODA is applied locally and globally to validate transactions, while in SESAMO, strict 2PL is applied locally [19] and globally [10]. Second, SESAMO does not elect cluster heads and does not apply 2PC; so it may randomly choose a server as the coordinating server.

5.1 Simulation Parameters and Performance Metrics

Our simulation models are implemented using the AweSim simulation language [20]. Global transactions are defined as entities, and mobile modes are defined as resources with different initial energy levels and randomly distributed locations.

The static simulation parameters and their values are shown in Table 1. We conducted experiments to study the impacts of inter-arrival time on the system performance, which is the mean of an exponentially distributed time between the arrivals of two consecutive transactions. The inter-arrival time is varied over the range from 1 to 10 seconds in order to vary the system load [21] and create a scenario with high data contention.

Table 1. Simulation Parameters

Static Parameters	Value	Reference
Simulation area (m^2)	1000x1000	[21]
Transmission range (m)	250	[22]
Node moving speed (m/s)	2	[22]
Total transactions	1000	[21]
Low Energy Threshold (LET)	50%	
Server energy consumption rate in active mode (watts)	30.3	[23]
Server energy consumption rate in doze mode (watts)	12.5	[23]

Five performance metrics are used and they are defined in Equations (5), (6), (7), (8), and (9), respectively: total time when servers are in active mode, abort rate, total number of cluster head reelections, total energy consumed by all servers, and average difference in remaining energy between two servers. Since transactions in mission-critical applications should be executed not only correctly but also within their deadlines, we use firm real-time transactions to evaluate the performance. 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).

A server is in active mode only if it is processing transactions; otherwise, it is in doze mode to save energy. The total time when servers are in active mode evaluates whether servers are busy to process transactions most of time, where m is the total number of servers and $T_{a,i}$ is the total time when server S_i is in active mode.

$$Total\ \ time\ when\ servers\ \ are\ \ in\ active\ mode\ = \sum_{i=1}^{m} T_{a,i} \tag{5}$$

The second performance metric is the abort rate to measure the percentage of aborted transactions, and can be computed as below:

$$Abort\ rate = \frac{Total\ \#\ of\ aborted\ \ transactions}{Total\ \#\ of\ generated\ \ transactions} * 100\% \tag{6}$$

The third performance metric is the total number of cluster head (primary and non-primary) reelections to evaluate whether an algorithm takes balancing energy among servers into consideration, where N_{primay} ($N_{non\text{-}primay}$) is the number of primary (non-

primary) cluster head reelections. However, more reelections do not guarantee more balanced energy among servers because there is an overhead of reelections and transferring the information from the old cluster head to the new one.

$$Total\ number\ of\ cluster\ head\ reelections = N_{primay} + N_{non-primary} \quad (7)$$

The fourth performance metric is the total amount of energy consumed by all servers in both active mode and doze mode. This metric evaluates how energy-efficient each technique is, where m is the total number of servers, ECR_a (ECR_d) is the energy consumption rate when a server is in active (doze) mode, and $T_{a,i}$ ($T_{d,i}$) is the total time when server S_i is in active (doze) mode.

$$Total\ energy\ consumed\ by\ servers = \sum_{i=1}^{m} (ECR_a * T_{a,i} + ECR_d * T_{d,i}) \quad (8)$$

The fifth performance metric is the average difference in remaining energy between two servers to evaluate how balanced the system is in terms of energy consumption. The more balanced the system is, the longer lifetime the system has. This metric is computed using the following formula, where m is the total number of servers, and RE_i and RE_j are the remaining energy of servers S_i and S_j, respectively.

$$Average\ difference\ in\ remaining\ energy\ between\ two\ servers = \frac{\sum_{i=1}^{m}\sum_{j=1}^{m}|RE_i - RE_j|}{(m-1)*m} \quad (9)$$

5.2 Simulation Results

Fig. 3 shows in both SODA and SESAMO, that the total time when servers are in active mode increases when the transaction inter-arrival time increases. The total time of SESAMO is always much longer than SODA's because SESAMO is pessimistic and uses locks to hold limited system resources to prevent conflicting transactions from accessing them. In other words, servers have to be in active mode most of time to keep processing transactions.

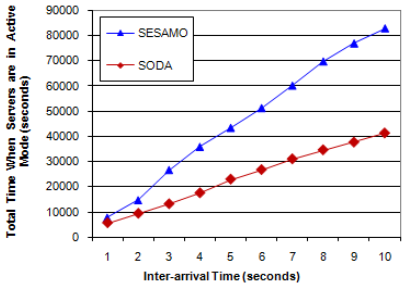

Fig. 3. The total time when servers are in active mode vs. inter-arrival time

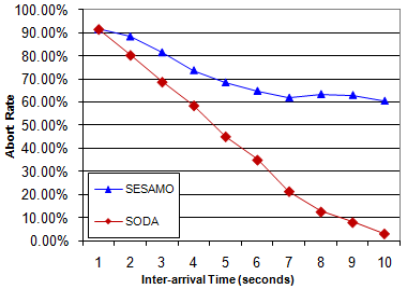

Fig. 4. The abort rate vs. inter-arrival time

Fig. 4 shows that the abort rates of SODA and SESAMO decrease when the transaction inter-arrival time increases. The abort rate of SODA is much lower than that of SESAMO right after the inter-arrival time is longer than 1 second. This is mainly because SODA is optimistic and non-blocking, and conflicts among transactions become rare, so that servers are not in active mode most of time (as shown in Fig. 3) and can process transactions in time. Although SESAMO does not enforce global serializability, strict 2PL running both locally and globally still blocks many conflicting transactions. When the inter-arrival time is getting shorter, it is easy to see that the abort rate of SODA is close to SESAMO's because conflicts among transactions increase; in addition, this confirms the fact that optimistic CC techniques work well only if conflicts among transactions are rare.

Fig. 5 shows the total number of cluster head reelections of SODA increases as the inter-arrival time increases. When the inter-arrival time reaches 10 seconds, the total simulation time is close to 3 hours (1000 transactions * 10 seconds = 10,000 seconds). Consequently, more cluster heads' remaining energy is below the predefined threshold *LET*, and more reelections are triggered to change roles for preserving energy. However, the total number of reelections of SESAMO is always zero because its design does not involve any cluster heads. In other words, SESAMO does not rotate roles among servers to balance energy.

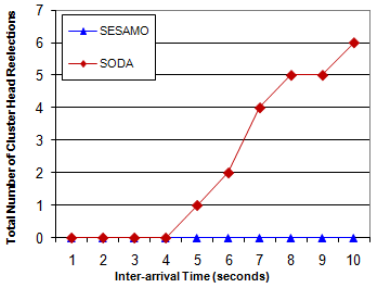

Fig. 5. The total number of cluster head reelections vs. inter-arrival time

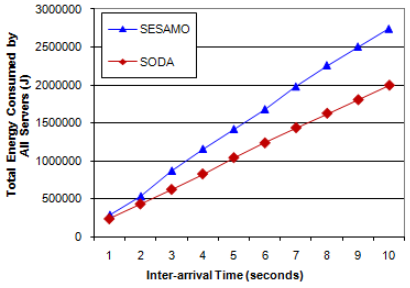

Fig. 6. The total energy consumed by all servers vs. inter-arrival time

Fig. 6 shows that the total energy consumption of all servers increases with the increase of the inter-arrival time. This is expected because more transactions are committed as inter-arrival time increases as shown in Fig. 4, so that each server has to spend more time in active mode on processing these committed transactions as shown in Fig. 3. SODA consumes at least 51,124 J and at most 749,727 J less than SESAMO right after when the inter-arrival time is longer than 1 second. This happens because transactions arrive into the system with a slow rate, and conflicts among transactions become much rarer, so that optimistic SODA performs better than pessimistic SESAMO due to no prevention of conflicts overhead.

The average difference in energy consumption between two servers when varying the inter-arrival time is shown in Fig. 7. Through this metric, we want to check whether the energy consumption is balanced among servers.

It is easy to see that SODA balances remaining energy better than SESAMO. This is because more non-primary cluster heads and primary cluster heads with higher energy are reelected as shown in Fig. 5. However, in SESAMO, there is no role rotation strategy and clients may keep submitting transactions to the same servers so that these servers are overloaded.

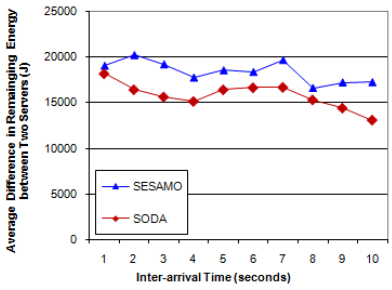

Fig. 7. The average difference in remaining energy between two servers vs. inter-arrival time

6 Conclusion and Future Research

In this paper, we introduced a database transaction concurrency control technique, called SODA, that can be used to support mission-critical applications such as disaster rescue and battlefields in a clustered MANET. This technique considers transaction real-time constraints as well as mobility, energy limitation, and workload of both mobile servers and mobile clients in a clustered MANET architecture. Our solution is aimed at reducing transaction abort rate while saving the energy consumption by servers and balancing the energy consumption among servers. With respect to these performance metrics, the simulation results show the superiority of SODA over the existing technique, SESAMO.

For future research, we plan to incorporate data replication into our model to improve data access time and availability. We will also investigate the impacts of mobility (speed) of mobile nodes, disconnection time and read-only transaction percentage on abort rate, response time, total energy consumed by all servers, and average difference in remaining energy between two servers. We also plan to investigate alternative commit protocols.

Acknowledgment: This material is based upon work supported by (while serving at) the National Science Foundation (NSF) and the NSF Grant No. IIS-0312746.

References

1. Alampalayam, S. P., Srinivasan, S.: 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 (2009)
2. Catarci, T., De Leoni, M., Marrella, A., Mecella, M., Salvatore, B., Vetere, G., Dustdar, S., Juszczyk, L., Manzoor, A., Truong, H.: Pervasive Software Environments for Supporting Disaster Responses. IEEE Internet Computing, vol. 12, no. 1, pp.26--37 (2008)

3. Lu, W., Seah, W. K. G., Peh, E. W. C., Ge, Y.: Communications Support for Disaster Recovery Operations using Hybrid Mobile Ad-Hoc Networks. In: Proceedings of the 32nd IEEE Conference on Local Computer Networks, Dublin, Ireland, pp. 763--770 (2007)
4. Silberschatz, A., Korth, H. F., Sudarshan, S.: Database Systems Concepts, McGraw-Hill College (2005)
5. Chlamtac, I., Conti, M., Liu, J.: Mobile Ad Hoc Networking: Imperatives and challenges. Ad Hoc Networks Publication, vol. 1, no. 1, pp. 13--64 (2003)
6. Sklavos, N., Touliou, K.: Power Consumption in Wireless Networks: Techniques & Optimizations. In: Proceedings of The IEEE Region 8, EUROCON 2007, International Conference on "Computer as a Tool" (2007)
7. Choi, M., Park, W., Kim, Y.: 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 (2009)
8. Lam, K., Wong, C. S., Leung, W.: Using Look-ahead Protocol for Mobile Data Broadcast. In: Proceedings of the 3rd International Conference on Information Technology and Applications, pp. 342--345 (2005)
9. Lei, X., Zhao, Y., Chen, S., Yuan, X.: Scheduling Real-Time Nested Transactions in Mobile Broadcast Environments. In: Proceedings of the 9th International Conference for Young Computer Scientists, pp. 1053--1058 (2008)
10. Brayner, A., Alencar, F. S.: A Semantic-serializability Based Fully-Distributed Concurrency Control Mechanism for Mobile Multi-database Systems. In: Proceedings of the 16th International Workshop on DEXA, pp. 1085--1089 (2005)
11. Xing, Z., Gruenwald, L., Phang, K. K.: A Robust Clustering Algorithm for Mobile Ad-hoc Networks. In: a chapter on the Handbook of Research on Next Generation Mobile Networks and Ubiquitous Computing. Editor Samuel Pierre, IGI Global, ISBN: 160566250X, pp. 187-200 (2010)
12. Wireless Ad Hoc Technology, http://www.atacwireless.com/adhoc.html
13. Chatterjee, M., Das, S. K., Turgut, D.: WCA: A Weighted Clustering Algorithm for Mobile Ad Hoc Networks. Cluster Computing, vol. 5, no. 2, pp. 193--204 (2002)
14. Basu, P., Khan, N., Little, T. D. C.: A Mobility Based Metric for Clustering in Mobile Ad Hoc Networks. In: Proceedings of IEEE ICDC, pp. 413--418 (2001)
15. Xue, M., ER, I., Seah, W. K. G.: Analysis of Clustering and Routing Overhead for Clustered Mobile Ad Hoc Networks. In: Proceedings of the 26th IEEE international Conference on Distributed Computing Systems, pp. 46--53 (2006)
16. Wang, Y., Kim, M. S.: Bandwidth-adaptive Clustering for Mobile Ad Hoc Networks. In: Proceedings of International Conference on Computer Communications and Networks, pp. 103--108 (2007)
17. Xing, Z., Gruenwald, L., Phang, K. K.: SODA: an Algorithm to Guarantee Correctness of Concurrent Transaction Execution in Mobile P2P Databases. In: Proceedings of the 19th International Conference on DEXA Workshop, pp. 337--341 (2008)
18. Hwang, S.: On Optimistic Methods for Mobile Transactions. Journal of Information Science and Engineering, vol. 16, no. 4, pp. 535--554 (2000)
19. Holanda, M., Brayner, A., Fialho, S.: Introducing self-adaptability into transaction processing. In: Proceedings of the 2008 ACM symposium on Applied computing, pp. 992--997 (2008)
20. Pritsker, A., O'Reilly, J.: Simulation with Visual SLAM and AweSim, 2nd edition. New York: John Wiley & Sons (1999)
21. Gruenwald, L., Banik, S. M., Lau, C. N.: Managing real-time database transactions in mobile ad-hoc networks. Distributed and Parallel Databases journal, vol. 22, no. 1, pp. 27--54 (2007)
22. Leu, Y., Hung, J. J., Lin, M. B.: A New Cache Invalidation and Searching Policy for mobile Ad Hoc Networks. In: Proceedings of the 2007 annual Conference on International Conference on Computer Engineering and Applications, pp. 337--343 (2007)
23. Notebookcheck, http://www.notebookcheck.net/Review-Lenovo-ThinkPad-T400s-Notebook.21081.0.html