Cache-Based Query Optimization In Mobile Ad-Hoc Networks

Kshama Raichura, Nilesh Padhariya, Kishor Atkotiya

Abstract: This work proposes query optimization model using caching for Mobile Ad-hoc Networks (MANETs). In the recent era, the proliferation of mobile devices increases the high traffic on Internet. Furthermore, the large number of smart-mobile devices is capable to carry considerable amount of data with them. Increasing storage capability of mobile devices has inspired us to perform effective query optimization on mobile database systems, where small part of the central database can be cached on local mobile database for performing faster query processing. This further reduces the need of constant connectivity of the device to the central server, which reduces the Internet traffic. In the query issuer retrieves data from the cached mobile database, which minimizes the traffic at central database server. The main contributions of our work are three-folds. First, we propose cache-based query optimization model for faster data retrieval and effective query processing. Second, we propose two schemes for cached-broker, namely Event-Driven Query caching (EDQc) and Location-Dependent Query caching (LDQc). Third, we conduct a performance study, which demonstrates that in order to retrieve data faster in mobile environment and to avoid large number of connection to the server, caching is one of the important concept.

Index Terms: Caching, Mobile Ad-hoc Network (MANET), Mobile Database System, Query Optimization

1 INTRODUCTION

THE usage of mobile devices are increasing day-by-day. Also the mobile devices are available with number of features like internet access, games and other applications means the usage of mobile devices are not limited just for communication purpose. These devices are functioning wirelessly and are able to form a mobile network to share the information with each other or with the central server. In mobile networks, energy-constrained devices are able to store data in form of databases, which are known as mobile databases. Moreover, the mobile user is able to retrieve the information from the other mobile device in the network. This process of information retrieval is initiated using mobile queries. Query processing and query optimization are the key factors that affect the overall performance, efficiency and reliability of the system. The key problem in query optimization in distributed environment is – how to select the most cost effective plan to execute a query. Efficient query processing is the basic requirement for any technology such as On-Line Analytical Processing (OLAP) and data mining and knowledge discovery in database. Our proposed model considers that the mobile device itself is capable to process small-scale queries on its own mobile databases.

- *Kshama Raichura, Shree M. & N. Virani Science College, Gujarat, INDIA, kdraichura@vsc.edu.in*
- *Nilesh Padhariya, Atmiya Institute of Technology & Science, Gujarat, INDIA, nilesh@aits.edu.in*
- *Kishor Atkotiya, J. H. Bhalodiya Women's College, Gujarat, INDIA, atkishor@yahoo.com*

Such queries are checked by the inbuilt parser and then the query optimizer evaluates and selects the best Query Execution Plan (QEP) for a given query. In mobile database system, the databases are resides in the cached memory of the mobile called cached-database. In our proposed architecture, the mobile node having a cached-database is known as *cached-broker.* The main contributions of our work are three folds.

1. We propose cache-based query optimization model for faster data retrieval and effective query processing.
2. We propose two schemes for cached-broker, namely Event-Driven Query caching and Location-Dependent Query caching.
3. We conduct a performance study, which demonstrates that in order to retrieve data faster in mobile environment and to avoid large number of connection to the server, caching is one of the important concept.

The reminder of this paper is organized as follows: Section 2 represents related work. Section 3 and Section 4 represent proposed architecture of cache-based query processing and caching schemes respectively. Performance study is given in Section 5 and in Section 6 we conclude our work with future direction.

2 RELATED WORK

A query is a language expression that describes data to be retrieved from database. Generally, the query may be defined as either (i) direct request for data by the user or (ii) a transaction that changes the stored data [1]. In query optimization context, it is often assumed that queries are expressed in a content-based manner. The total cost of any query optimization is the sum of communication cost, secondary access storage cost, computational cost & query latency.

2.1 Distributed Environment

The hardest problem in the distributed database area is query optimization. A distributed database is a collection of independent cooperating centralized systems [2]. There is a need of communication network to transfer data for the query processing in distributed database. There is particular cost of every query execution plan, here query execution plan means a query optimization method and cost is the summation of local cost and data transferring cost between sites. As the

number of relations in query is increased, it increases the complexity and cost. Moreover, less amount of data transmission is also an important factor to minimize the query processing cost and also a single query can be executed in several different ways. The work in [3] focuses a novel approach to reduce the volume of data transferred in the network by introducing reduction filters. The [4] proposes one new query optimization algorithm based on multi-relation semi joins, which reduces the volume of intermediate result and effectively decreases the overall cost of network communication. The autonomy and heterogeneity of component databases give rise to a number of issues, which make MQO (Multidatabase Query Optimization) a distinct problem from DQO (Distributed Query Optimization) [5]. The work in [6] describes the distributed heterogeneous database structure of query optimization system based on the Hibernate technical for the B/S structure and analysis the calculation of the response time. The work in [7] proposes an agent-based technology using mobile query optimization for distributed data warehouse and OLAP applications. The content in [8] shows the evaluation of cost model and on the basis of that model agents may choose the appropriate execution site. Moreover, the work in [9] represents the compile-time placement method of Mobile Relational Operators MROs in a large scale environment. Generally, the mobile peers do not have a perfect knowledge of its position and the ambiguity of position is not considered to select the control access: this is the reason behind the failure of conventional controllers such as collisions [10]. The work in [11] presents how integration of single query optimization, multi-query optimization and cache investment is helpful in query execution in distributed database systems.

2.2 Mobile Environment

Recently mobile database technology is widely used in various fields including information sharing, near filed communication, weather forecasting, healthcare, mobile office, M-learning, stock activities and public services. For the mobile database system, the emergence of Mobile Ad-hoc Network (MANET) brings innovative organization. The model based on mobile agents provides an execution strategy over the network for each query is proposed in [12]. The reaction of mobile agents in case of errors and resource unavailability is shown in [13]. Aggregation is an important tool for the data reduction. The application of generic query interface for data aggregation on the ad-hoc network is shown in [14]. As collision is one of the common reason of energy consumption, the main aim of [15] is generation of query trees to minimize the energy consumption per query and collisions in concurrent data transmissions. The [16] also represents optimization algorithm of middle result, query separation algorithm, query execution nodes selection algorithm and query plan execution algorithm. The framework named MULS (Multi-Level Service) is combination of OLS (an algorithm) and SCI (the procedure) designed for the information retrieval system from multiple items in sequential order is proposed in [17]. By considering the features of the mobile devices, the [18] proposed an agent based mobile database model. This model mixed up all the members, decompose the long-lived transaction for the effective query optimization of mobile database and improve the efficiency of data processing in M-learning.

3 ARCHITECTURE

This section discusses the proposed architecture of cached-based query optimization model for mobile database system.

3.1 Query Processing

This section discusses the steps involved in query processing starting from formation of query to execution including query optimization.

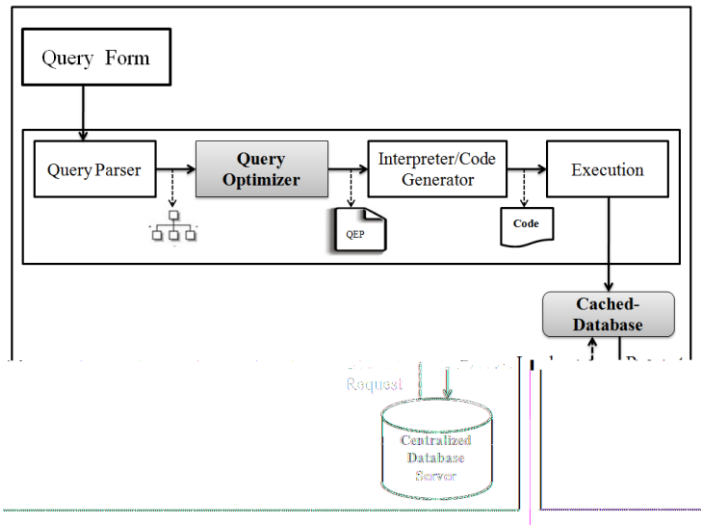

Fig. 1. Query Processing

Figure 1 represents steps involved in query processing. Each step is described as below:

Query Form: Queries are medium to discover various types of information from the database. Generally, the queries are formed in some specific language like SQL. Mainly these queries are formed either for Data Retrieval purpose or to perform Transaction on the data. Once the query has been formed, the processing of query starts.

Query Parser: The query parser validates the query in terms of syntax, attributes and relations. After validating the query, it will translate the query into an internal form using relational calculus.

Query Optimizer: The role of query optimizer is to find out the Query Execution Plan (QEP) for the available query. The query optimizer selects the cheapest QEP to execute the query.

Interpreter / Code Generator: Interpreter transforms optimizer generated QEP into calls to the query processor. The code generator generates code for the selected QEP. According to QEP, the interpreter generates executable code for *Q*.

Execution: On the completion of all the previous steps, the query processor executes the code for *Q*.

Cached-Database: In our proposed model, the query is processed using the data available on cached-database of other mobile devices. This cached-database holds small partition of database into local memory to provide the result faster to the query-issuer *QI*. The holder of the cached-

database is known as cached-broker. It also requests centralized database server for the unavailable data.

Centralized Database Server: The centralized database server holds the whole (main) database. It provides data to the cached-brokers on their request.

3.2 Query Optimization Schemes

In our previous work [19], we proposed two query optimization schemes namely, Sequential Query Optimization (SEQ) and Sorted Query Optimization (SRT). In sequential optimization method, the *Q* is going to execute in a sequential manner from left to right. The SEQ retrieves data by following the order of columns appear in *Q*. While sorted optimization method, first sorts the columns appear in *Q* and then retrieves data. Here the difference between these two schemes is the time duration within which a particular query completes the execution. Prioritizing the columns using sorting technique will improve the speed of data retrieval and the whole execution process will take less time as compare to Sequential Optimization method.

3.3 Query-based Caching

This section discusses the cache-based query optimization in Ad-hoc Networks. Storing small partition of centralized data into local cache memory of mobile device is called Caching. The concept of caching plays an important role to improve the response time of query. Frequent disconnection, limited processing capacity, small display, battery powered, etc. are the limitations of the mobile devices but the users of these devices demand faster data retrieval at minimum cost. Caching also helps to minimize the communication cost and traffic at centralized database server.

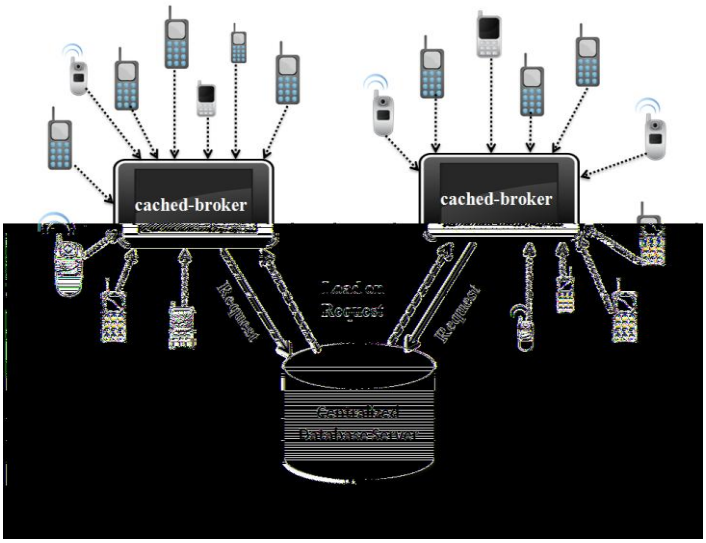

Fig. 2. Query-based Caching

Figure 2 represents the caching operation for query processing in Query-based Caching for effective query optimization.

In query-based caching, the main database is stored on centralized database server. When *query-issuer QI* issues the query *Q*, the *Q* will transfer to the neighboring node. The neighboring node first checks the cached-database, if data available on cached-database satisfy the *Q*, then node will respond to the *QI* with resultant data. In case of data unavailability, the node will ask for the same to the centralized database server, acquires data from the server, stores this result in the cache memory of its own and then respond to the *QI*. The node, which acquire maximum number of *Q*, stays for a long time (maximum time) in connected mode and has capacity to store data in a cache memory, will play the role of cached-broker. The cached-broker manages its cached-database on the basis of FAQs (Frequently Asked Queries). On the basis of these FAQs, the cached-broker will collect broad view of data from centralized database servers and then give specific data as a result of any *Q* raised by *QI*. In this way, the cached-database is updated frequently and has of course small amount of specific data. The cached-broker helps *QI* to retrieve efficient data faster and minimizes traffic at centralized database server and communication cost.

4 Caching Schemes for Optimization

This section discusses the two methods of caching in mobile database system. We designate these schemes as Event-Driven Query caching (EDQc) and Location-Dependent Query caching (LDQc). In both of the schemes, the cached-broker will retrieve the data in a broad form based on Frequently Asked Query (FAQ) and then serve the data to the query-issuer *QI* on demand. Both schemes minimize traffic on server.

4.1 Event-Driven Query caching (EDQc)

In Event-Driven Query caching, the cached-broker cache the data on the basis of Frequently Asked Queries (FAQs) during past specific *time*. For example, assume that today is the day of result. The students are continuously hammering on the university website to get their result. In such situations there are chances of network jam on university server. The result of this is: students are not able to get their result on time. In this case, if one of the mobile devices will hold this data related to result and plays the role of cached-broker, then it will reduce the traffic at the centralized database server at university and the students will get their results on time. For example, consider a *query Q* as a combination of set of tables, columns and constraints, $Q(T_{set}, C_{set}, D_{set})$. Where, T_{set} is a set of tables, C_{set} is a set of columns and D_{set} is a set of constraints involved in *Q*. The cached-broker stores the data related to the FAQs rise in during past specific time *t*.

$$EDQc = \sum_{q=0}^{NQ} (t) \tag{1}$$

Where *t* is a time period considered by cached-broker to cache the data for FAQs and N_Q is the number of queries raised during particular *t*.

4.2 Location-Dependent Query caching (LDQc)

In Location-Dependent Query caching, the cached-broker cached the data on the basis of FAQs by referring the current location. For e.g. vehicles on the high way require data like: “nearest petrol pump from the current location” or “nearest restaurant from the current location”. So, to provide information faster to this on road QI, the cached-broker will store the data related to locations on road. For example, consider a query *Q* as a combination of set of tables, columns and constraints, $Q(T_{set}, C_{set}, D_{set})$. Where T_{set} is a set of tables,

C_{set} is a set of columns and D_{set} is a set of constraints involved in *Q*. The cached-broker stores the data related to the FAQs raised for a particular location l.

$$LDQc = \sum_{q=0}^{NQ} (l) \quad (2)$$

Where l is the location for which the *Q* is raised and N_Q is the number of queries considered by cached-broker for data caching.

4.3 Illustrative Examples of EDQc and LDQc

Figure 3 depicts the illustrative example of cache-based query optimization using EDQc and LDQc. For Example, centralized database server has set of tables (Table A, Table B, Table C, Table D and Table E) and from that cached-broker has Table B, Table D and Table E as cached-database on the basis of FAQs.

Example 1: Consider a formed query Q, which requires data from *Table A, Table C and Table D*. When cached-broker receives this *Q*, it first checks the cached-database. From the given *Tset* by *Q*, the cached-database has only one table that is *Table D*. The cached-broker has to select data of remaining two tables from the centralized database server. When cached-broker request for the data of these two tables, the operation on the data is performed by the Centralized Database Server then the cached-broker gets the data. After retrieving data from the centralized database server, the cached-broker performs cross product of these two datasets: first data set means *Table D* and second data set means the cross product of *Table A and Table C*, which is return by server. After doing all these process, the cached-broker is able to give response to the *query-issuer Q* with requested data. When *QI* requests the data from both, local databases and centralized database at that time, communication cost will be higher because in these types of *Q*, first the *QI* requests to the cached-broker and then cached-broker requests to the Centralized Database Server, which will double the communication. In this case the cached-broker will execute only partial query.

Example 2: Consider a formed query *Q*, which requires data from *Table B*, *Table D* and *Table E*. When cached-broker receives this *Q*, it first checks the cached-database. From the given *Tset* by Q, the cached-database has all the tables. In this case the cached-broker performs the cross product of all the Tables of *Tset*. And then give response to the *QI* with requested data. When *QI* requests for the data, which is available on local database, in this case the cached-broker is able to execute whole *Q*, because the cached-broker has all that tables of *Tset*, which are involved in to generate the result of *Q*. This will minimize the communication cost because for the processing of these types of *Q*, the cached-broker needs not to connect with the centralized database server.

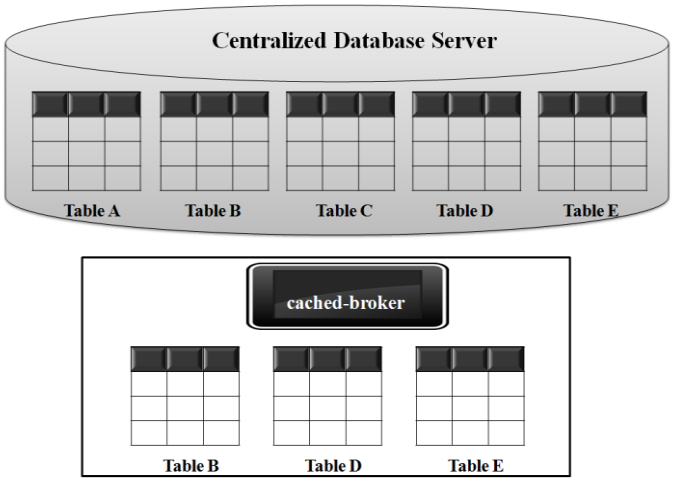

Fig. 3. Illustrative Example of EDQc and LDQc

5 Performance Study

This section represents the performance study of these two caching methods. In Event-Driven Query caching (EDQc), the cached-broker caches the data related to particular event. So, the main parameter that affects the caching is time. In Location-Dependent Query caching (LDQc), the cached-broker cached the data related various locations. So, in this method the main parameter that affects the caching is *location*. In both the methods, the cached-broker stores the data related to Frequently Asked Queries (FAQs). The result of this caching is faster response time because queries are answered by cached-broker instead of centralized server.

TABLE 1
PARAMETERS OF PERFORMANCE STUDY

Parameter	Value
Number of MPs	100
Processor of MPs	1GHz to1.9 GHz
Size of Cached Database	10MB to 25MB
RAM	256MB to 1GB
Probability of MP availability	50% to 75%

In our experiments, we consider mobile devices having the processor range from 1 GHz to 1.9 GHz. The default value of storage (RAM) is 256 MB. Each mobile device contains data base of minimum 20MB, which varies up to 100MB. Here the performance metrics are average response time (ART), CPU process time (CPT) and messages (MSG). We define a query as **completed** if the query issuer *QI* receives data within 70% of the query deadline time *TQ*. We compute ART only for the completed queries.

$$ART = \frac{1}{NC} \sum_{q=1}^{NC} (tb - te) \quad (3)$$

Where *tb* is the query-issuing time, *te* is the time of the query result reaching the query-issuer, and *NC* is the total number of

completed queries. CPT is the time required by the CPU to process *Q*. We compute CPT only for the completed queries.

$$CPT = \frac{1}{NC}\sum_{q=1}^{NC}(tcp) \quad (4)$$

Where *tcp* is the processing time by CPU in seconds to process and *NC* is the total number of completed queries. CMT is the time duration that *Q* takes to reach from source to destination and vice versa. We divide *Q* into sub parts according to available operations in it. Each sub part is denoted as MSG. Thus,

$$MSG = \sum_{q=1}^{NQ} mq \quad (5)$$

Where *mq* is the number of messages incurred for the q[th] query.

5.1 Performance of EDQc and LDQc

We conducted this experiment using default values in Table 1. Figure 3 depicts the results. As the Number of Queries is increased, ART, CPT and MSG are increasing.

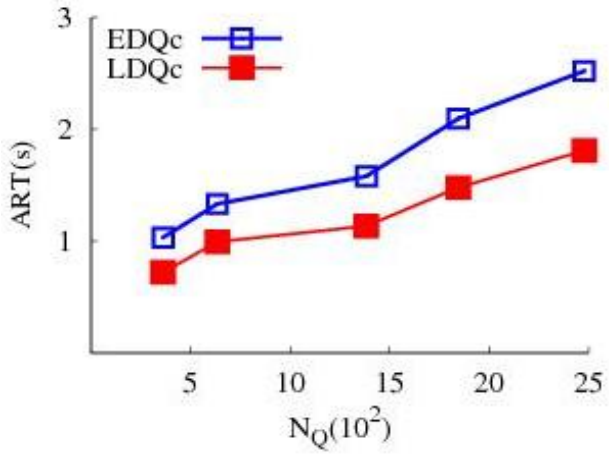

Fig. 3(a). ART

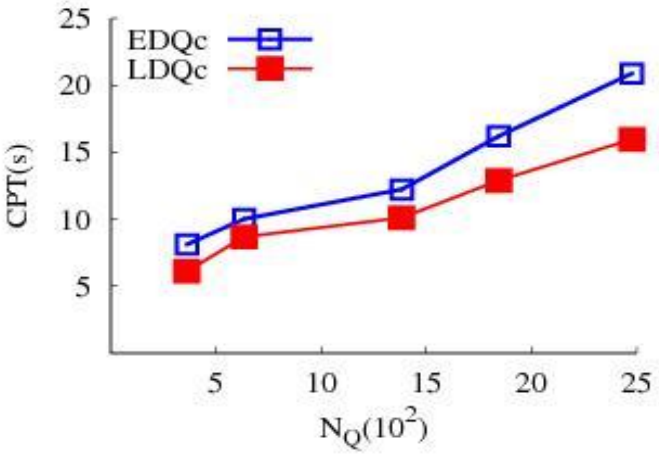

Fig. 3(b). CPT

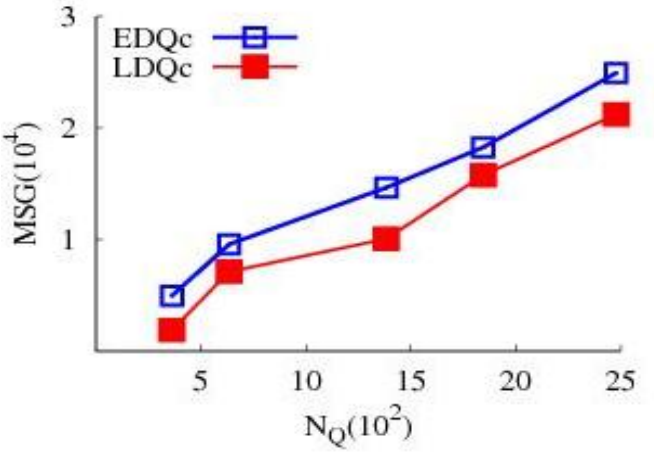

Fig. 3(c). MSG

Fig. 3. Performance of EDQc and LDQc

During the processing of 800 to 1200 queries ART and CPT are stable because of enough data availability at cached-broker related to recent FAQs. The size of Cached- Database is affected by the number of FAQs. Increasing number of FAQs will increase the Cached-Database. It is more time consuming to process a query from large database as compare to small sized database. After performing 1500 queries, the size of Cached-Database is going to increase and this is the reason behind increasing ART and CPT after 1500 queries.

5.2 Effect of variations in Cache size (C_Z)

Figure 4 represents the effect of variations in cache size (C_Z). Increasing ART in initial stage indicates small cache size (C_Z). In initial stage the cached-database is constructing by analyzing frequent FAQs. After collecting data related to recent FAQs, the cached-database has enough data to answer a query. Moreover, answering a query from cached-database reduces the communication cost and this turns into minimum ART. As the cached-database is going to increase, it will increase ART. Searching data from large database is a time consuming process, which turns into high ART. CPT increases sharply for all the variations because query processing using large database requires more computation time. MSG is decreasing sharply during the C_Z is between 20MB to 100MB. When cached-broker gets maximum amount of data from the centralized database server, then there is no need to generate query to the centralized database. Increasing C_Z indicates minimum number of MSG generation and increasing size of C_Z will decrease the number of transactions with centralized database server. The less number of transactions means less number of MSG.

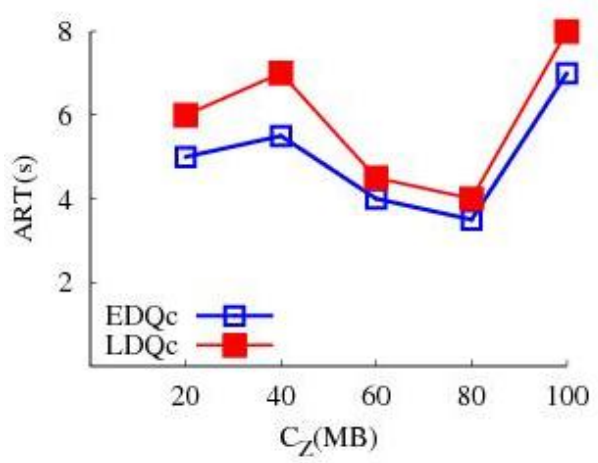

Fig. 4(a). ART

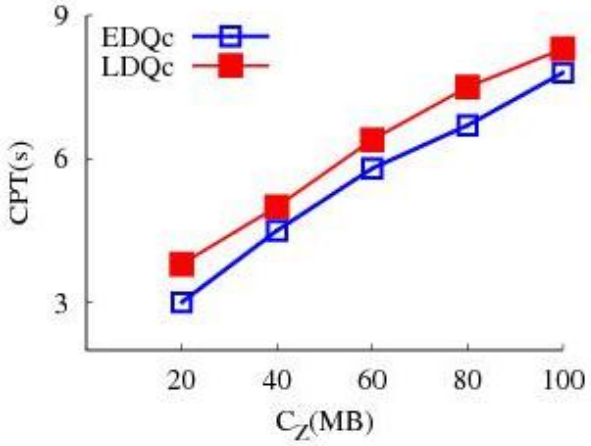

Fig. 4(b). CPT

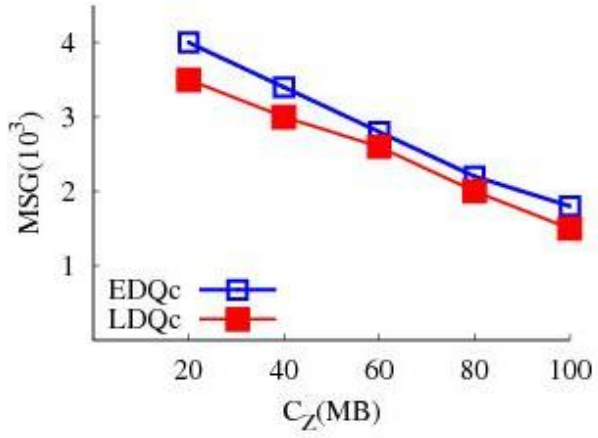

Fig. 4(c). MSG

Fig. 4. Effect of variations in Cached-database (C_Z)

5.3 Effect of variations in number of mobile peers (N_{MP})

Figure 5 depicts the effect of variations in number of mobile peers (N_{MP}). As N_{MP} increases, ART, CPT and MSG increases for all the approaches. Increment in number of mobile peers (N_{MP}) will result in high volume of query density. Moreover, the high volume of query density turned into high volume of MSG. Data caching related to FAQs is the affecting factor for the stability of CPT while processing queries of 400 to 800 mobile peers (N_{MP}). Processing of queries using cached-data reduces the connections with centralized server, which turns into low communication cost.

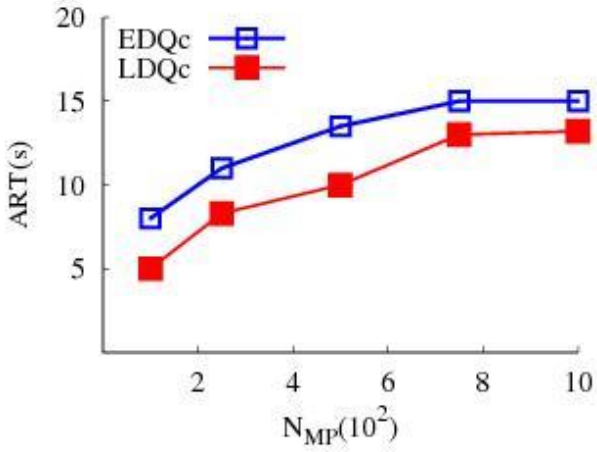

Fig. 5(a). ART

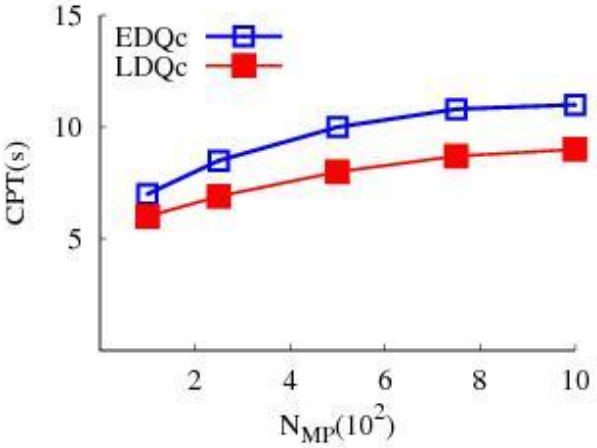

Fig. 5(b). CPT

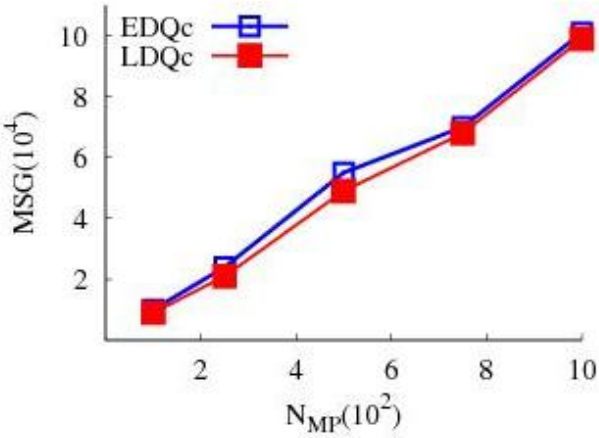

Fig. 5(c).MSG

Fig. 5. Effect of variations in number of mobile peers (N_{MP})

6 Conclusion

We have proposed the cache-based query optimization model for MANETs through our proposed caching schemes, namely Event-Driven Query caching (EDQc) and Location-Dependent Query caching (LDQc). In our proposed approach, the query issuer retrieves data from the cached-database, which minimizes the traffic at central database server. In Event-Driven Query caching, cached-broker cached the data by considering recent FAQs in last specific time while in Location-Dependent Query caching, the cached-broker cached the data by considering recent FAQs related to location. Here in both the schemes the cached-broker retrieves data in broad view from the server and then serves the data to the query issuers on demand.

References

[1] M. Jarke and J. Koch, "Query optimization in database systems," ACM Computing Surveys, vol. 16, pp. 111–152, 1984.

[2] P. Doshi and V. Raisinghani, "Review of dynamic query optimization strategies in distributed database," in Electronics Computer Technology (ICECT), 2011 3rd International Conference on, vol. 6, 2011, pp. 145–149.

[3] J. Morrissey, "Reduction filters for minimizing data transfers in distributed query optimization," in Electrical and Computer Engineering, 1996. Canadian Conference on, vol. 1, 1996, pp. 198–201 vol.1.

[4] X. Li, D. Li, H. Z. Gao, and L. Yao, "Study of query of distributed database based on relation semi join," in Computer Design and Applications (ICCDA), 2010 International Conference on, vol. 1, 2010, pp. V1–134–V1–137.

[5] H. Lu, B.-C. Ooi, and C.-H. Goh, "Multidatabase query optimization: issues and solutions," in Research Issues in Data Engineering, 1993: Interoperability in Multidatabase Systems, 1993. Proceedings RIDE-IMS '93., Third International Workshop on, 1993, pp. 137–143.

[6] Z. Zhenyou, L. Bin, and C. Zhi, "The research on the query optimization on the distributed heterogeneous database based on the response time," in Computer Science and Network Technology (ICCSNT), 2011 International Conference on, vol. 3, 2011, pp. 1541–1544.

[7] A. Hameurlain and F. Morvan, "Mobile query optimization based on agent-technology for distributed data warehouse and olap applications," in Database and Expert Systems Applications, 2002. Proceedings. 13th International Workshop on, 2002, pp. 795–799.

[8] M. Hussein, F. Morvan, and A. Hameurlain, "Embedded cost model in mobile agents for large scale query optimization," in Parallel and Distributed Computing, 2005. ISPDC 2005. The 4th International Symposium on, 2005, pp. 199–206.

[9] B. Ergenc, F. Morvan, and A. Hameurlain, "Robust placement of mobile relational operators for large scale distributed query optimization," in Parallel and Distributed Computing, Applications and Technologies, 2007. PDCAT '07. Eighth International Conference on, 2007, pp. 227–235.

[10] V. A. Huynh and N. Roy, "iclqg: Combining local and global optimization for control in information space," in Robotics and Automation, 2009. ICRA '09. IEEE International Conference on, 2009, pp. 2851–2858.

[11] I. Azari, "Efficient execution of query in distributed database systems," in Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on, vol. 4, 2010, pp. V4–428–V4–433.

[12] T. Win and K. M. L. Tun, "Mobile agent cooperation methods in hybrid query optimization," in Information and Telecommunication Technologies, 2005. APSITT 2005 Proceedings. 6th Asia-Pacific Symposium on, 2005, pp. 71–76.

[13] A. Hameurlain and F. Morvan, "Invited paper: How can mobile agents and cost model approaches helpful for distributed query optimization," in Database and Expert Systems Applications, 2006. DEXA '06. 17th International Workshop on, 2006, pp. 617–621.

[14] S. Madden, R. Szewczyk, M. Franklin, and D. Culler, "Supporting aggregate queries over ad-hoc wireless sensor networks," in Mobile Computing Systems and Applications, 2002. Proceedings Fourth IEEE Workshop on, 2002, pp. 49–58.

[15] V. Zadorozhny, P. Chrysanthis, and A. Labrinidis, "Algebraic optimization of data delivery patterns in mobile sensor networks," in Database and Expert Systems Applications, 2004.
Proceedings. 15th International Workshop on, 2004, pp. 668–672.

[16] H. Ke, "Two-phase query optimization in mobile adhoc wireless networks," in Intelligent Computing and Intelligent Systems, 2009. ICIS 2009. IEEE International Conference on, vol. 3, 2009, pp. 515–520.

[17] H.-P. Hung and M.-S. Chen, "Muls: A general framework of providing multilevel service quality in sequential data broadcasting," Knowledge and Data Engineering, IEEE Transactions on, vol. 19, no. 10, pp. 1433–1447, 2007.

[18] J. Wang, "Research on data transmission and storage in field of m-learning," in Information Engineering and Computer Science, 2009. ICIECS 2009. International Conference on, 2009, pp. 1–4.

[19] K. Raichura, N. Padhariya, and K. Atkotiya, "QoMoD: Effective Query Optimization in Mobile Database Systems," Journal of Information and Communication Technology, vol. 3, no. 9, pp. 9–17, 2013.