DELite: Database Support for Embedded Lightweight Devices

Krithi Ramamritham and Rajkumar Sen
Indian Institute of Technology, Bombay
Mumbai-400016, India
{krithi}@iitb.ac.in

Computation platforms have extended to small intelligent devices like cellphones, sensors, smartcards, PDAs, etc. As new functionalities and features are being added to these devices, increasing number of applications are being developed, many of them dealing with significant amounts of data leading to the need of embedded database support on these devices [1]. The queries go beyond simple Select-Project-Join queries but still have to be locally executed on the device [4]. Most of the modern day cellphones are being equipped with increasing memory which means more data centric applications are being developed for them. Sensor networks are also proliferating and these collect data from the environment and subject them to various queries. Most of these queries need to be executed on the device itself to reduce communication costs. Applications for PDAs execute complicated join and aggregate queries on the device resident data. Thus, there is an increasing need to facilitate the execution of complex queries locally on a variety of lightweight computing devices.

However, scaling down the database footprint poses challenges since these lightweight devices come with very limited computing resources. While the amount of main memory and stable storage available in such devices is relatively small, the devices are not uniformly endowed with resources. For example, the computing capability and main memory of a cellphone differs from that of a PDA. It is essential that the available resources be utilized optimally for a database system that is developed for such devices. The already limited stable storage has to accommodate the operating system as well as the database system code, which means even less storage is available to store the data.

Storage Models designed for such database systems should reduce storage cost to a minimum to be able to store more data. Limited stable storage usually precludes the creation and use of any additional index structures, hence the storage models should try to incorporate some index information in the data model itself. Ideally, index structures which can speed up query processing at no additional storage cost should be maintained. Different storage models have different storage and update costs. The selection of the best storage model for a data attribute in a relation depends on the size of the relation, selectivity and length of the attribute, frequency of updates, and the nature of queries.

As far as the choice of query processing techniques is concerned, RAM is perhaps the most critical resource in these devices. Existing approaches use minimum memory algorithms for every operator. This can lead to poor performance for complex queries involving several joins and aggregates. Query execution time for complex queries can be reduced by using operator algorithms which build in-memory indices and save the aggregate values. Query plans should be generated in such a way that they make effective use of the available memory. Optimal memory allocation among the database operators is a must if we need to ensure the best usage of memory. Another factor that influences query execution is the storage model used to store the relations. Each storage scheme entails a different cost for selecting and projecting a tuple, hence the cost of a query execution plan will vary across storage schemes. Thus, the selection of a query execution plan for a handheld device should be governed by (i) the amount of memory available, and (ii) the underlying storage model. Memory and storage model cognizant query plan generation is hence essential. Also, the query optimizer itself should not be too complex to be executed on the handheld device.

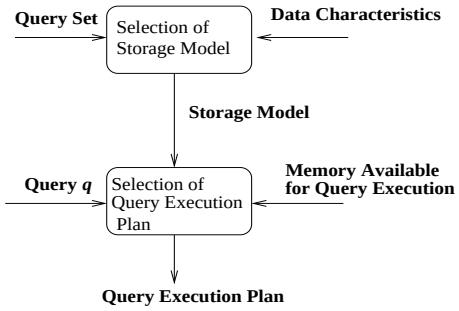

Figure 1: Selection of storage scheme and query execution plan

Figure 1 characterizes the requirements of the database system. The system should select the storage model based on data characteristics and the expected type of queries. For any query q, depending on the storage model and amount of available memory, the optimal query processing technique should be determined.

Lightweight versions of some of the popular DBMSs like Oracle Lite, Sybase Adaptive Server Anywhere, IBM DB2 EveryPlace, and Microsoft SQL Server for Windows CE have been developed with a reduced code footprint by stripping down database features. IBM DB2 EveryPlace organizes data in a flat way; records are stored sequentially and column values are stored contiguously in the record. The query optimizer does not generate and examine different evaluation plans for a query. Only nested loop algorithm is used to evaluate joins.

Instead of having a statically defined storage scheme and query processing technique independent of the database application and the device resources, we need to have a dynamic approach where we choose the storage model and query processing schemes depending on the application characteristics and the resources present

EMSOFT'04, September 27–29, 2004, Pisa, Italy.
ACM 1-58113-860-1/04/0009.

in the device. There has been some work to this effect. For example, [4] proposes a new storage model called Ring Storage for smartcards that combines data and index storage in a single structure. Their use of ring indices produces a negative side effect on query execution since selections become costly. The query processing techniques proposed are quite specific to the smartcard platform where the amount of memory available is extremely small. The optimizations suggested for aggregate queries rely on the underlying storage and index model.

We need to design algorithms that can cater to the needs and constraints of various type of devices. The non-uniformity of resources across handheld devices can however result in a need for a special purpose database system for each type of device, e.g. smartcards, cellphones, PDAs. An approach that can cater to the needs of different types of lightweight computing devices is needed:

1. In order to reduce the storage cost new storage techniques may be needed and the best storage scheme should be selected based on data characteristics and nature of queries.

2. The query processing engine should optimally allocate the available memory among the operators. It should choose the minimum cost query plan for a given handheld device depending on the amount of available memory and the underlying storage scheme.

The DELite project at IIT Bombay is aimed at addressing these issues and makes the following contributions:

- We have utilized a novel storage model, ID based Storage, which is an improvement over the existing Domain Storage Model [3]. For many data intensive small device applications, ID Storage wins over Domain Storage and thus reduces storage costs considerably. It also provides a unidirectional Join Index between a foreign key-primary key pair thus speeding up joins. We examine the suitability of this scheme vis-a-vis existing storage schemes and depending on the data and query characteristics, select the best storage scheme for an attribute.

- Given the need for optimal allocation of memory among the database operators based on the cost function of the operator algorithms, we have shown that the exact memory allocation algorithm proposed in [5] can be modified to be used in our context. However, since the time and space complexity of the exact algorithm can prevent it from being used in some devices, we have also developed a heuristic solution with a reduced complexity based on the benefit of an operator per unit memory allocation.

We have implemented our techniques on the Simputer [2], a handheld device, and our experience with this implementation indicates that (a) ID based Storage can lead to lower storage cost for the dataset used in [4] and (b) Our query processing techniques based on the memory allocation algorithms always select the query plan with minimum cost and so, memory and storage model cognizant query optimization is both feasible and essential.

Our approach [6] is essentially one that composes a DBMS- its storage schemes and query processing techniques- in a device and DBMS application conscious fashion rather than in a single-size-fits-all manner. To take this approach forward, in ongoing work, we are building a modular DBMS toolkit for handheld devices. Modules from this toolkit can be plugged into a system depending on the type of the application and resources of the device.

1. REFERENCES

[1] Small Databases are Beautiful, Database Trends and Applications, August 2003. http://www.dbta.com.

[2] The Simputer. http://www.simputer.org.

[3] A. Ammann, M. Hanrahan, and R. Krishnamurthy. Design of a Memory Resident DBMS. In *IEEE COMPCON*, 1985.

[4] C. Bobineau, L. Bouganim, P. Pucheral, and P. Valduriez. PicoDBMS: Scaling down Database Techniques for the Smartcard. In *VLDB*, 2000.

[5] A. Hulgeri, S. Sudarshan, and S. Seshadri. Memory Cognizant Query Optimization. In *COMAD*, 2000.

[6] Rajkumar Sen and Krithi Ramamritham. Efficient data management on lightweight computing devices. June 2004. http://www.cse.iitb.ac.in/ krithi/papers/??.pdf.