DATA CACHING ON MOBILE DEVICES
The Experimental MyMIDP Caching Framework

Hagen Höpfner and Sebastian Wendland and Essam Mansour
International University in Germany, Campus 3, 76646 Bruchsal, Germany
hoepfner@acm.org, sebastian.wendland@i-u.de, essam.mansour@acm.org

Keywords: Mobile Information Systems, Data Caching, MIDP, Database Connectivity, MySQL

Abstract: Data caching is an appropriate technique for reducing wireless data transmissions in mobile information systems. The literature describes numerous caching approaches on a theoretical level. Semantic, preemptive, or context aware caches are discussed but not implemented for mobile devices even evaluated in real applications. The problem here is the complexity of data management tools. Software for mobile devices must consider the limited footprint of the used hardware as well as the restrictions of the application programming interfaces. In this poster paper we describe the caching framework of our MyMIDP database driver. The framework provides interfaces that allow to implement the caching approaches discussed in the literature and to test them on MIDP 2.0 enabled mobile devices in a MySQL environment. Hence, we provide researchers with a necessary tool for proofing the efficiency of their caches.

1 INTRODUCTION

Nowadays data access is almost possible everywhere and at any time. Mobile information systems (mIS) use wireless links for data communication. Current technologies like UMTS or wireless LAN provide high transmission rates, and flat rate contracts reduce the monetary costs. However, wireless data transmission is energy intensive, and using it reduces the up-time of mobile devices drastically. One solution to this problem is to cache data once it has been received. The literature on mIS focuses on semantic caching. At this, query results are completely stored and indexed using the corresponding query (Keller and Basu, 1996; Lee et al., 1999; Ren and Dunham, 1999). This allows for analyzing the cache content. In the best case new queries can be answered without communication with the server. To a certain degree it is also possible to supplement cached data to answer a query (Godfrey and Gryz, 1999; Höpfner and Sattler, 2003; Ren and Dunham, 2003). In addition to this semantic caches even more enhanced approaches have been researched in the context of mIS. Preemptive caches, a technology that is also known as "data hoarding", (Peissig, 2004; Kubach and Rothermel, 2001; Liu et al., 1996) use query rewriting techniques in order to cache data before it is explicitly used. This increases the probability of re-usability of the cache content. Context aware caches (Ren and Dunham, 2000; Zheng et al., 2002) take the context of the usage of the system into account when deciding upon caching and/or replacing data.

However, almost all of these ideas were discussed but not implemented for mobile devices or even evaluated in real applications. The challenge addressed in this paper was to provide researchers with a tool for proofing the efficiency of their caches while considering the limited footprint of mobile devices and the restrictions of the available application programming interfaces. Taking this into account, we developed a caching framework for implementing various caching approaches. It is an extension to our MyMIDP (Höpfner et al., 2009a; Höpfner et al., 2009b) driver that allows for directly accessing MySQL databases from MIDP 2.0 enabled mobile devices.

The remainder of this poster paper is structured as follows: Section 2 presents the caching extension of the MyMIDP driver. Section 3 illustrates the usage of the caching framework. Section 4 concludes this short paper and gives an outlook on future research.

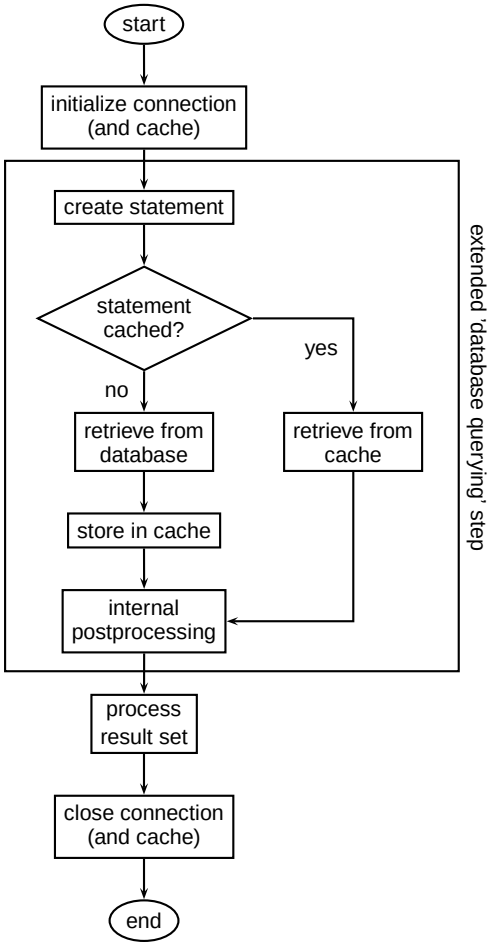

Figure 1: Cache framework general workflow

2 THE MYMIDP CACHE EXTENSION

The MyMIDP cache extension transparently extends the MyMIDP driver with a caching system. It provides local data caching for database query result sets, thus reducing the mobile device's dependency on unreliable and low bandwidth connections for data transfer and, therefore, saving energy.

Here, transparency means that an application does not notice the difference between a cached and a not cached connection, or the difference between a result set from the database and one from the cache. As the workflow in Figure 1 shows, most of the caching logic is hidden in the normal "database querying" step and is thus not visible to the user.

Overall, the caching workflow works as follows: If the result set of a SQL statement is found in the cache, it is loaded, processed and returned to the user without the need of any communication with the database server. Otherwise, the statement is executed and its result set cached. On default the driver uses a per-statement caching approach with a ten entry cache and a first-in, first-out (FIFO) replacement strategy.

All cached data is kept in MIDP's record management system (RMS), a nonvolatile collection of records of binary data. A database result set is serialized into a set of records (one record for each row in the result set) and written to a record store. A central cache handler is responsible for tracking the result sets and can retrieve them as required. There are two reasons for this approach. First, the MIDP standard does not define a file-access API. Second, mobile devices only have a few hundred kilobytes of main memory, prohibiting an in-memory cache.

Listing 1: FIFO Cache Replacement

```
public interface CacheStrategy
{
    public String processQuery(String query)
            throws SQLException;
    public String cacheMiss(String query) throws
            SQLException;
    public CachedResultSet postProcessQuery(String
             query, CachedResultSet result)
            throws SQLException;
}
```

Listing 2: FIFO Cache Replacement

```
public interface CacheReplacementStrategy
{
    public Vector sessionStart(Enumeration
            cacheContent);
    public CacheEntry newEntry(Enumeration
            cacheContent, CacheEntry newEntry);
    public boolean sessionEnd();
}
```

3 CACHE CUSTOMIZATION

The driver defaults to a simple per-statement caching with a FIFO queue. However, the framework provides two extension points for application programmers to change this default behavior. Utilizing them allows to implement more enhanced caching approaches as discussed in Section 1.

The *first* extension point provides a way to customize the caching strategy – the way the driver decides what to store in the cache. The *second* one deals with cache replacement – the way the cache is kept up-to-date. Both extension points are implemented as Java interfaces (cf. Listings 1 and 2) and are set during the driver's initialization phase.

3.1 Cache Strategy

The cache strategy controls the way the driver decides what to store in the cache. This is accomplished via three lifecycle methods which are called during the execution of an SQL statement. These methods roughly correspond to the stages in the workflow shown in Figure 1.

Before a statement is executed, the driver scans through the cache index to check, whether the statement is already cached or not. This is preceded by a call to the `processQuery` method, which determines the cache index key from the SQL statement.

If the search for the key comes up empty, the `cacheMiss` method is called. The task of this method is to modify the original SQL statement into the SQL statement required[1] by the caching strategy, which is then send to the MySQL server instead of the original statement. The produces result set is then added to the cache, using the cache key provided by the `processQuery` method.

After the driver finished creating a result set, either from cache or from the remote database, it is post-processed by the `postProcessQuery` method. The task of this method is to create a result set that is identical to the one which would have been created without the use of caching, thus ensuring transparency. The new result set is then passed to the application.

A preemptive cache example: To show the capabilities of this interface, lets have a look at how the included preemptive caching strategy is implemented. It uses the idea of expanding the column selections of the SQL statement to include all columns, leaving the other clauses untouched (i.e. a `SELECT a, b, c FROM xyz` becomes a `SELECT * FROM xyz`).

First, the `processQuery` method uses the `FROM` (and following `WHERE`, `GROUP BY ...`) clause as cache index key. It simply cuts the front of SQL statement off and returns the part beginning at the `FROM` identifier (i.e. a `SELECT a, b, c FROM xyz` becomes `xyz`). When the entry is not found, the `cacheMiss` method simply takes the result of the `processQuery` method and adds a `SELECT *` to its beginning, thus creating a valid SQL query which is then send to the database server. Finally, the `postProcessQuery` method creates a new result set based on the original query. For each column definition in the original SQL statement the corresponding column from the database result set is selected and added to the new one. Afterwards, eventual renaming is applied where necessarry (as defined by SQL `AS` clauses). Finally, the new result set is returned to the client.

[1] e.g. for enabling preemptive caching

Of course, this kind of caching can fail quite easily, for example with the use of SQL functions or joins. However, the application developer knows the properties of the strategy used in his application and can thus write the SQL statements in a more robust way. Also, there is always the option of not using caching for some statements. Furthermore, most problems can be avoided with a more intelligent SQL analysis. However, please note that the current version of MIDP does not implement any regular expression engine, a developer must hence implement his own, a comprehensive undertaking in it's own.

3.2 Cache Replacement Strategy

The cache replacement strategy controls the actual content of the cache. This is done via the three methods listed in Listing 2. The `sessionStart` and `sessionEnd` methods are both only called once during the drivers lifecycle, namely during the initialization and termination phases, respectively. The `newEntry` method on the other hand is called every time the cache content is to be modified.

During the initialization phase of the driver, the cache content index is read from the RMS by the cache manager and is then passed to the `sessionStart` method of the cache replacement strategy. This method identifies the obsolete cache entries and returns them in order to be deleted. Similarly, the `sessionEnd` method is called during driver shutdown, indicating whether or not the cache should be cleared or not. Overall, the job of these two methods is to move the cache into a valid state. For instance, because the default FIFO strategy indicates a cache flush during driver shutdown, it also marks the entire cache content for deletion during initialization.

The task of the `newEntry` method, on the other hand, is to manage the cache content. It is called whenever a new entry is to be added to the cache and identifies at most one cache entry which is to be removed from the cache. The default FIFO strategy, for instance, does return the oldest cache entry once the cache reached a predefined size of ten entries.

4 CONCLUSIONS AND OUTLOOK

This paper presented our caching framework for mobile devices. It is an extension for the MyMIDP database driver for MIDP 2.0 compatible devices. As mentioned in Section 1 we had to consider device and API limitations during the design and development process. The given goals were reached. Our current

development version of the MyMIDP driver (incl. the described cache implementations) has a footprint of 27kB, only. The caching framework provides flexible and transparent data caching for database result set.

For testing purposes we implemented only one straight forward caching strategy. However, more enhanced approaches like semantic, preemptive or context aware caches can also be implemented with our framework. Therefore, future areas of research include also the implementation of a MIDP 2.0 compatible SQL analysis library that is necessary for analyzing queries. Also, given the current rapid growth in mobile device capabilities, a memory based cache will sooner or later move into the area of feasibility.

The final goal is to provide a database framework similar to the capabilities of the standard Java implementation (i.e. Java SE).

Note: The GPL lizensed MyMIDP source code as well as our proof-of-concept MySQL client for mobile phones are available online at `http://it.i-u.de/dbis/myMIDP`.

REFERENCES

Godfrey, P. and Gryz, J. (1999). Answering queries by semantic caches. In Bench-Capon, T., Soda, G., and Tjoa, A. M., editors, *Database and Expert Systems Applications: Proceedings of the 10th International Conference, DEXA'99*, volume 1677 of *LNCS*, pages 485–498, Heidelberg. Springer-Verlag.

Höpfner, H. and Sattler, K.-U. (2003). Towards Trie-Based Query Caching in Mobile DBS. In König-Ries, B., Klein, M., and Obreiter, P., editors, *Post-Proceedings of the Workshop Scalability, Persistence, Transactions - Database Mechanisms for Mobile Applications*, number P-43 in LNI, pages 106–121. Gesellschaft für Informatik, Köllen Druck+Verlag GmbH.

Höpfner, H., Schad, J., Wendland, S., and Mansour, E. (2009a). MyMIDP: An JDBC driver for accessing MySQL from mobile devices. In Chen, Q., Cuzzocrea, A., Hara, T., Hunt, E., and Popescu, M., editors, *Proceedings of the The First International Conference on Advances in Databases, Knowledge, and Data Applications (DBKDA 2009), March 1-6, 2009, Cancun, Mexico*, pages 74–80. IEEE Computer Society.

Höpfner, H., Schad, J., Wendland, S., and Mansour, E. (2009b). MyMIDP and MyMIDP-Client: Direct Access to MySQL Databases from Cell Phones (Demo). In Freytag, J.-C., Ruf, T., Lehner, W., and Vossen, G., editors, *Proceedings of the 13. Conference on Business, Technology, and Web, March 2-6, 2009, Münster, Germany*, volume P-144 of *Lecture Notes in Informatics (LNI) - Proceedings*, pages 604–607, Bonn, Germany. Gesellschaft für Informatik, Köllen Druck+Verlag GmbH.

Keller, A. M. and Basu, J. (1996). A predicate-based caching scheme for client-server database architectures. *The VLDB Journal*, 5(1):35–47.

Kubach, U. and Rothermel, K. (2001). Exploiting location information for infostation-based hoarding. In Rose, C., editor, *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, pages 15–27. SIGMOBILE, ACM Press.

Lee, K. C. K., Leong, H. V., and Si, A. (1999). Semantic query caching in a mobile environment. *ACM SIGMOBILE Mobile Computing and Communications Review*, 3(2):28–36.

Liu, G., Marlevi, A., and Maguire, G. (1996). A mobile virtual-distributed system architecture for supporting wireless mobile computing and communications. *Wireless Networks*, 2(1):77–86.

Peissig, J. (2004). guidePort – An Information and Guidance System. In Kyamakya, K., editor, *WPNC 04 Proceedings*, number 0.1 in Hannoversche Beiträge zur Nachrichtentechnik, pages 1–17, Aachen. NICCIMON, IEEE, VDI, Shaker Verlag GmbH.

Ren, Q. and Dunham, M. H. (1999). Using clustering for effective management of a semantic cache in mobile computing. In Banerjee, S., Chrysanthis, P. K., and Pitoura, E., editors, *Proceedings of the 1st ACM International Workshop on Data Engineering for Wireless and Mobile Access*, pages 94–101, New York, NY, USA. ACM Press.

Ren, Q. and Dunham, M. H. (2000). Using semantic caching to manage location dependent data in mobile computing. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 210–221. ACM.

Ren, Q. and Dunham, M. H. (2003). Semantic Caching and Query Processing. *Transactions on Knowledge and Data Engineering*, 15(1):192–210.

Zheng, B., Xu, J., and Lee, D. (2002). Cache Invalidation and Replacement Strategies for Location-Dependent Data in Mobile Environments. *IEEE Transactions on Computers*, pages 1141–1153.