Towards Providing Complete Knowledge in Constraint-based Database Caching

Martin Tritschler, Volker Hudlet

Databases and Information Systems Group
Dept. of Computer Science, University of Kaiserslautern
P.O.Box 3049, 67653 Kaiserslautern, Germany
{m_trit, v_hudlet}@informatik.uni-kl.de

Form of work: study project
Advisor: Prof. Dr. Theo Härder
GI department: Database and Information Systems

Abstract: Database caching accelerates access to frequently requested data of a remote database (backend) by keeping subsets of records close to applications. Thereby, the constraint-based approach primarily uses constraints to determine what predicate extensions are complete and which queries can be answered. Equipping the backend with so-called *complete knowledge* about the caches' configurations enables it to make sufficient predictions about the contents stored in the caches. Furthermore, complete knowledge obviates the unnecessarily high communication costs caused by the currently supported simpler *structural knowledge* during update propagation. In addition, with complete knowledge it becomes possible to prepare the loading of contents on backend side. This paper describes the basic requirements and proposes an index structure that enables complete knowledge on the backend.

1 Motivation

Database caching accelerates access to data of a remote database system by providing frequently requested data close to applications. Using constraint-based database caching (CbDBC), a cache stores sets of records which represent predicate extensions whose consistency is controlled and preserved by constraints. This enables a dynamic adaptation of the cache contents to the evolving query workload, a behavior that cannot be assured by approaches based on full-table caching or materialized views [LGZ04]. In CbDBC one or more caches are connected to the primary copy which is managed by the backend. It performs all database updates and propagates the needed changes to the cache instances. Correct and efficient change propagation is an essential prerequisite for implementing a global transaction control.

Without further assistance the backend cannot predict the subset of caches affected by a certain change operation. It is therefore unable to avoid propagation to all caches, which in turn provokes an unnecessary communication overhead. In order to achieve a targeted

propagation the backend needs to maintain information about the state of its connected caches. We will show in section 3 that by providing the backend with *complete knowledge* (metadata about the cache which allows to reconstruct its exact contents) it is enabled to perform a precise preselection of change propagations, which leads to a reduced communication load. However, to keep this metadata consistent with the caches' loading and unloading behavior, it has to be accessed and modified very frequently. Therefore, we will propose an index structure in section 4 which enables efficient look-up and maintenance of this metadata. The following section will explain the required basics of CbDBC.

2 Principles of Constraint-based Database Caching

A CbDBC system caches records of predicate extensions from backend tables in related, equally structured cache tables. A backend table T has a cache table T_{C} assigned, which only contains a logical subset of its data. Furthermore, the cache manages two kinds of constraints: *filling columns* (FC) and *referential cache constraints* (RCC). Both constraints are defined using the concept of *value completeness*: The value v in a column $T_{\mathrm{C}}.a$ is *value complete* if and only if all records of the related backend table T containing v are present in the cache table T_{C} [HB07]. The set of all cache tables and constraints is called *cache group*.

An RCC $S_{\mathrm{C}}.s \rightarrow T_{\mathrm{C}}.t$ defines a constraint in a cache group between two cache table columns, where the column $S_{\mathrm{C}}.s$ is called source column and the column $T_{\mathrm{C}}.t$ is the target column. An RCC is satisfied if and only if all values v in the source column are value complete in the target column. So a value v appearing in the source column implies value completeness of v in the target column [Bra08].

Filling columns control the process of loading values into the cache. A query referencing in a filling column $T_{\mathrm{C}}.f$ a value v, which is a member of the set of loadable values K, triggers the loading of v to make it value complete in $T_{\mathrm{C}}.f$. If $T_{\mathrm{C}}.f$ has outgoing RCCs, the loading of records needs to be recursively continued to preserve all affected constraints [BHM06, KBM09].

3 Advantage of complete knowledge

The ACCache project has implemented a middleware-based CbDBC prototype [BHM06]. For consistency control a primary-copy strategy is used in which all writing operations take place in the backend. A component called *Write Set Manager* captures all changes which occur in the backend database within a transaction and maintains information about them in a data structure called *Write Set (WS)*, which is used for the pinpointed propagation to the affected cache instances [THK09].

Since a cache only stores a subset of the backend data, propagation of all changes to all caches (and consequently to unaffected caches) would cause substantial, but unnecessary

communications. If the backend just knows the definition of each cache group (*structural knowledge*), it can only perform a coarse-grained selection of records. For both update and delete operations, false positives can occur whenever a certain cache contains an affected table, but no affected records for it. To enable a more fine-grained and accurate preselection, the backend needs additional knowledge about the records stored in each cache. To provide efficient look-up of these extended metadata (complete knowledge), the backend maintains an index structure which can deliver for every modified record an indication of the caches storing it. Thus, for a given WS, all affected caches can be detected.

Moreover, the additional task of handling the loading and unloading behavior of caches [KBM09] can benefit from this index structure. The index structure can be used to determine the set of records which have to be loaded or unloaded. Given a certain RCC and its source column, it is easy to identify the records that have to be present in the target column due to value completeness. Therefore, RCC metadata has to be included in the index structure, which will be shown in the remainder of this paper.

4 Implementation of complete knowledge

As seen in the previous section, complete knowledge is used to identify the exact set of caches which are affected by the data manipulation operations contained in a WS. This task is supported by a hash-based index structure, which allows fast and direct access to the requested metadata. In order to avoid duplication of records, the implementation only indexes row identifiers (RowIDs) provided by the underlying DBMS. For each table, a separate index is maintained.

Figure 1a gives an example where, for the given table name CUSTOMER and the given RowID 3 (both are provided by the WS), all caches containing this record are identified. In this example, cache 1 and cache 2 are affected and so the change information will only be propagated to these two caches instead of all. The determination of the record set to be loaded in order to satisfy the RCC requirements—the second task mentioned in the

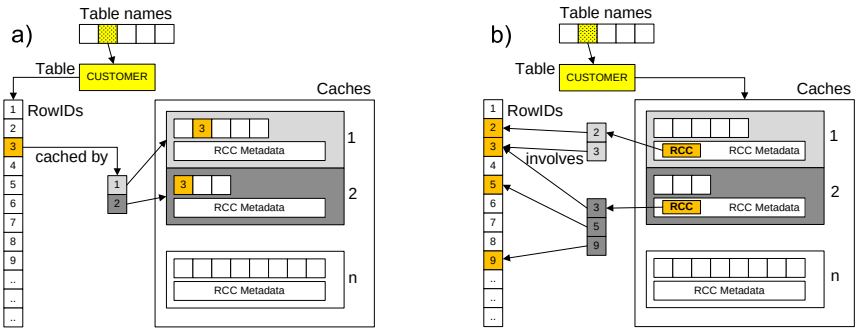

Figure 1: Sample situations of index use: a) identification of caches affected; b) determination of an RCC-based record set

previous section—uses the same index structure in a reverse way. Figure 1b illustrates this scenario: For the given table name CUSTOMER and cache 1, all RowIDs, which are implied by an RCC, are found.

A particular challenge is to keep this structure consistent with the cache contents. Whenever a cache instance unloads a subset of records, this information needs to be propagated to the backend. In this case of unloading, the adjustment of the index can be handled in a lazy way, because a temporary inconsistency only results in an unnecessary communication overhead as mentioned before. In the case of loading, however, this adjustment has to be applied immediately, because the backend must ensure that all changes needed in a specific cache instance are delivered in time to guarantee correct cache-evaluated query results.

5 Conclusion

As we have shown, complete knowledge avoids unnecessary communication. The presented index structure allows the identification of the exact set of affected caches for each change recorded in a WS. By using this index, the change propagation can be performed in an efficient way, which is an important basis for implementing a global transaction control mechanism. Additionally, the index structure helps to select the exact subset of records that needs to be loaded into the cache at the backend side. A mechanism called *prepared loading* [KBM09] can be implemented more efficiently.

It still has to be explored whether the additional administrative overhead caused by the index maintenance is justified by the saved communication overhead. Therefore, a performance analysis evaluating the benefits of different knowledge levels at the backend side will be subject of future research.

References

[BHM06] Andreas Bühmann, Theo Härder, and Christian Merker. A Middleware-Based Approach to Database Caching. In *Proc. ADBIS*, LNCS 4152, pages 182–199, Thessaloniki, 2006.

[Bra08] Susanne Braun. Implementierung und Analyse von Synchronisationsverfahren für das CbDBC. Master's thesis, Technische Universität Kaiserslautern, 2008.

[HB07] Theo Härder and Andreas Bühmann. Value Complete, Column Complete, Predicate Complete – Magic Words Driving the Design of Cache Groups. *VLDB Journal*, 17(2):805–826, 2007.

[KBM09] Joachim Klein, Susanne Braun, and Gustavo Machado. Selektives Laden und Entladen von Prädikatsextensionen beim Constraint-basierten Datenbank-Caching. In *Proc. BTW*, 2009.

[LGZ04] Per-Åke Larson, Jonathan Goldstein, and Jingren Zhou. MTCache: Transparent Mid-Tier Database Caching in SQL Server. In *Proc. ICDE*, pages 177–189. IEEE, 2004.

[THK09] Martin Tritschler, Volker Hudlet, and Joachim Klein. Aspekte der Konsistenzsicherung beim Constraint-basierten Datenbank-Caching. In *Proc. BTW, Stud.-Prog*, 2009.