Extending Data Warehouses by Semi-Consistent Database Views

Lutz Schlesinger, Wolfgang Lehner

University of Erlangen-Nuremberg (Database Systems)
Martensstr. 3, Erlangen, 91058, Germany
EMail: {schlesinger, lehner}@informatik.uni-erlangen.de

Abstract

The extremely expensive process of integrating data to achieve a consolidated database is one of the main problems in building a data warehouse. Since operational data sources may exhibit a high update rate, the data warehouse information is out of date by the order of magnitude. To avoid out-dated information the alternative is to query the data sources directly, which results in higher query runtimes or in the complete failure producing a result if data sources are currently not available. This paper discusses an approach to close this gap: multiple snapshots of participating data sources are cached in a middleware layer and user queries are routed to this set of snapshots which are providing an almost consistent global database view. We describe the architecture of our information middleware approach, develop different join semantics to combine different data sources, and propose algorithms for picking time consistent cuts in the history of local snapshots.

1. Introduction

Data warehouse databases are integrated and time-variant ([Inmo96]), but usually not up-to-date. In contrast data sources of data warehouse environments may be volatile, not integrated and time-invariant. Since the management of any industrial company is interested in the most current and consistent data, a possible solution might be the following: If consistent data are requested, the data warehouse is queried, although the data are not up-to-date. If the most up-to-date information is required, external information is extracted on-the-fly, integrated and combined to produce the result set for a single incoming query. This method yields a result consisting of the most current information. However, the processing of the query takes much time or might be even impossible if a data source is currently unreachable. To avoid the distinction, an additional integration layer is needed. From a structural point of view, data from external sources could be partially prefetched or the result of data warehouse queries could be cached to enable and/or speed up future queries. From a operational point of view, the middleware service has to provide a technology to transparently produce a semi-consistent view over the cached information. An approach to exploit the existence of temporalized materialized snapshots, is the topic of the paper. Thus, the overall idea presented in this paper is to provide a framework to combine data valid at different points in time so that the global consisteny is violated as little as possible.

The critical notion of consistency in the context of this work is defined from the following three perspectives:

- *existential strategy:* A precondition of a consistent global database view in web information systems is that no information which should be part of the global result is getting lost during the combination of two data sets relying on regular inner join semantics. Since we focus on this existential consistency strategy, we propose a technique of consistency bands in combination with different join semantics to come up with a semi-consistent global view over the participating data souces.
- *projection strategy:* Combining data sources in a union style may lead to duplicate entries. To eliminate duplicates, which might not be there in a fully consistent world, a second consistency strategy is used to apply aggregation operators to quantitatively express the degree of un-consistency.
- *scalar strategy:* A third perspective in defining consistency quantitatively appears when combining entries from different data sources using scalar operations. We express the inconsistency by a range of possible correct values framed by a minimum and maximum.

Summarizing the main conflict between timeliness and consistency leads to the idea of providing a general consistency framework with adjustable parameters to reflect the balance between up-to-date information and efficient query processing.

The presented conflict can also be found in two well known system architectures: (1) The database middleware approach (e.g. Garlic [IBM01b]) provides a direct access to all registered data sources, which are connected by wrappers to the middleware software. (2) The data warehouse approach ([Inmo96], [Kimb96]) periodically performs bulk loads into a single centralized database (figure 1). From a user point of view, the middleware approach enables query capabilities for a broad range of data sources. The query answers reflect the current state of each local data source. However, combining local data sources may yield an inconsistent global view ([ShLa90]). The other extreme may be reached following the data warehouse approach, where only historic states of the data sources are accessible by the users; new data remains hidden until the next data warehouse refresh. Permanent updates would cause tremendous effort in maintaining by simultaneously blocking users accessing the data warehouse in an analyzing way. Closing this gap between offering instant access to new or modified data and providing an almost consistent view

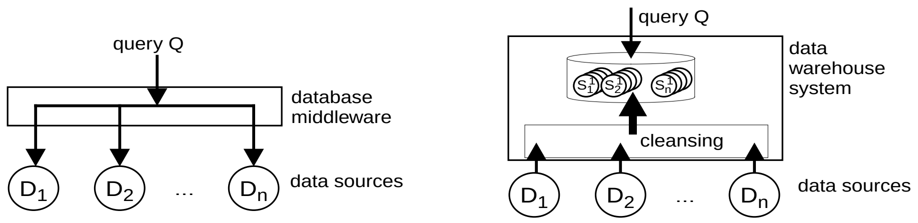


Fig. 1: Comparison of database middleware and data warehouse architecture

with regard to the global database is the focus of this paper. Our approach is to temporarily store multiple snapshots of the single data sources and provide mechanisms to pick those snapshots for a user query which provide the database view with the highest level of consistency.

Contribution and Organization of the Paper

In the remainder of this paper, we focus only on the actuality problem, the data consistency problem is not addressed. Therefore, we first sketch the overall architecture. Then we introduce the concepts of a historic cut, of consistency bands and join semantics as a means to develop algorithms for semi-consistent database views. Finally different strategies to generate historic cuts are outlined. Section 5 proposes an extended version of a historic cut picking algorithm and outlines the architecture of the prototype. The paper closes with a summary and conclusion.

Related Work

Since our approach is related to many different research areas, we classify our material as follows:

•*web information systems*
Web information systems (WIS) integrate Web-based systems and non-Web-based systems such as database and transaction processing systems ([IsBV98]) to provide a transparant view over a set of independent and (at least technically) heterogenous data sources. WIS emphasize the integration at the data level in contrast to 'Enterprise Application Integration' ([Lint99]).

•*mediator and database middleware systems / multidatabase systems*
Database middleware systems like DataJoiner/Garlic ([IBM01a], [IBM01b]) or mediator systems in general ([Wied92], e.g. TSIMMIS [IBM01c]) provide a query engine with sophisticated wrapper and query optimization techniques ([RoSc97]). Since those systems including multidatabase systems ([BrGS92], [ShLa90]) are only considering the current state of a data source, the integration point of different states of data sources from a global point of view is not a subject of interest.

•*materialized views in data warehouse systems*
One of the main optimization techniques in data warehouse systems is the use of materialized views. Since materialized views do not necessarily have to be consistent with the base data, browsing a database, where a single query is rerouted to multiple materialized views reflecting different states of the underlying database, yields a situation which is strongly related to our problem. Although a vast amount of work is done in the area of materialized views (selecting: [GHRU97]; maintaining: [GuMu99]; rewriting: [ZCL+00]), we are not aware of any work pinpointing our problem of generating semi-consistent database views. Moreover, it is important to mention that the proposed local caching strategy has nothing to do with incremental updates because the data are stored as a snapshot of the data source.

•*replication and concurrency control in database systems*
Introducing replicas ([ÖzVa91]) is a general way to speed up local access to data. Since replicas are synchronized immediately in the context of an update operation, our approach is much more related to the concept of database snapshots ([AdLi80]). The selection of the "best" set of snapshots from a consistency point of view, however, has not been discussed prior to our work.

2. Architecture

Before being able to introduce our approach of constructing a semi-consistent database view, we introduce the concept of a "bubble graph" (BG). A bubble graph may be visualized as a matrix, holding snapshots for different data sources taken at different points in time. Figure 2 shows a BG with a single snapshot for data source D_1, two snapshots for D_2 and three snapshots taken at different times for D_n. Snapshot S_n^3 does reflect the current state of D_n, because it was taken at t_{NOW}, the current time.

Using the technique of bubble graphs, we may display the idea of database middleware and the data warehouse approach. In the middleware approach a state of different snapshots only exists at t_{NOW} due to the absence of multiple snapshots for a single data source (figure 3 (a)). In contrast each set of snapshots in a data warehouse which is loaded at the same time is consistent due to the cleansing and transformation procedure. Therefore, the snapshots of the same loading period form a horizontal line (figure 3 (b)).

Mixing and adding flexibility to these approaches yields our architectural proposal, where each data source may be represented by an arbitrary number of snapshots (at least one) taken at *any* time without synchronizing with other data sources (figure 4). A query from the users will be routed not directly to the data source but to those snapshots providing the most consistent view, i.e. exhibit the minimal distance to a horizontal line in a BG.

The main advantage of this approach is the high flexibility when dealing with data. Adding a new snapshot to the system may be performed at any time and without blocking users reading data. Physical optimization like index generation or precomput-

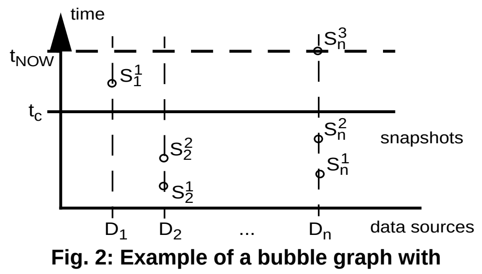


Fig. 2: Example of a bubble graph with multiple snapshots for multiple data sources

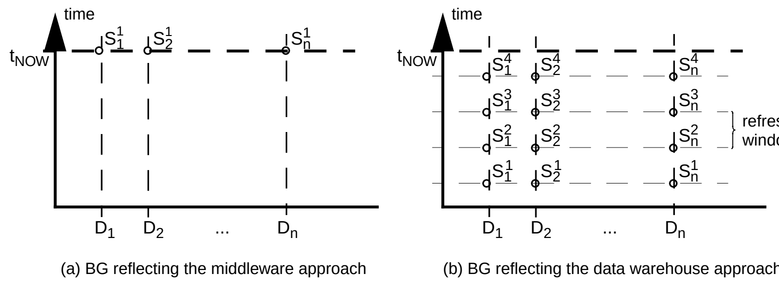

time
t_{NOW}
S_1^1
S_2^1
S_n^1
D_1
D_2
...
D_n
(a) BG reflecting the middleware approach
time
t_{NOW}
refresh windows
(b) BG reflecting the data warehouse approach

Fig. 3: Bubble graphs for middleware and data warehouse approach

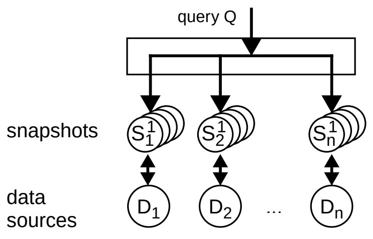


Fig. 4: Snapshot-based architecture

ing summary values may be done offline after taking the snapshots and before making the data accessible to the user. However, the extreme flexibility must be compensated by algorithms dealing with resulting inconsistencies between snapshots representing different states of the modeled world.

3. Historic Cuts, Consistency Bands, and Join Semantics

From the example presented in the last section, we are in the position to argue that a view over multiple data sources is consistent if all snapshots are placed on a horizontal line within a bubble graph*. Such a horizontal line is called a 'historic cut' and is denoted by t_c. Consistency from a database point of view addresses the maintenance of local constraints like checking primary key or check constraints and preserving referential integrity between two data sources. Since the first class of constraints must be checked by the data sources locally, it is required from our approach to focus on relationships between data sources and maintaining the join semantics. However, holding on tightly to the 'classical' and strict notion of consistency, we are not able to achieve any benefit in comparison to regular information system architectures. By relaxing the notion of consistency a bit, we may take advantage of the flexible architecture with independent snapshots.

Join Semantics

To reflect the possible inconsistencies arising by combining snapshots taken at different points in time, we have to propose certain repair mechanisms. To explain it on an example in figure 5 for a given historic cut t_c, only S_3^2 and S_4^2 are directly taken at t_c, so that we may combine them using regular inner join semantics, i.e. $S_3^2 \bowtie S_4^2$. All other snapshots have to be added to this consistency island formed by S_3^2 and S_4^2 using outer join semantics so that no information from the consistency island is lost. Consider the example from figure 5 with the tuples attached to the snapshots participating in a global database view: Obviously, $S_3^2 \bowtie S_4^2$ results in $\{(a_1,b_1,c_1),(a_2,b_2,c_2)\}$. Adding S_n^2 with inner join semantics yields a loss of (a_2,b_2,c_2) because this snapshot does not yet exhibit the corresponding join partner (c_2,d_2) which appears in S_n^3 . Hence, applying outer-join semantics for snapshots outside of the consistency island preserves as much information as possible. In our example it would result in a left-outer-join, i.e. $S_4^2 ⟕ S_n^2 = \{(b_1,c_1,d_1), (b_2,c_2,\text{NULL})\}$.

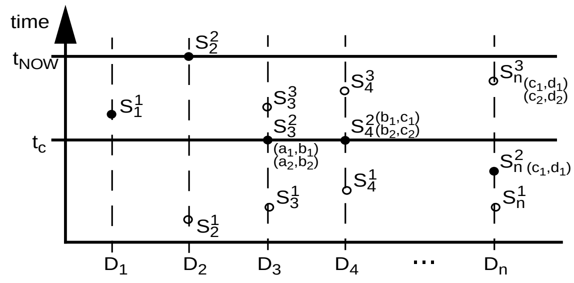


Fig. 5: Example for historic cuts and join semantics

* At this point, we may annotate that we are not dealing with schema intregration or instance transformation. These tasks are regarded a prerequisite already in directly querying the data sources.

Inner and Outer Consistency Bands

Referring again to the example from figure 5, we may detect two 'flaws' in the current procedure of combining independent snapshots from different data sources. First of all, a snapshot may be too old to contribute to a more-or-less consistent database view, e.g. S_2^1 . Therefore, we introduce an outer consistency band denoting the maximal distance from a snapshot to the current historic cut. Figure 6 presents a similar scenario compared to figure 5 with the corresponding outer consistency band. If no snapshot of a queried data source is within this band, we demand a snapshot refresh ([AdLi80], [GuMu99]). If the snapshot allows to be incrementally 'rolled forward' ([LHM+86], [SBCL00]) to any specific point in time, we distinguish two strategies. The *exact* mode rolls the snapshots to t_c; the *fuzzy* mode rolls it to the upper border of the outer consistency band so that it might still be added using outer join semantics. However, if the underlying data source does not provide any means for incrementally maintaining the snapshots, we have no other choice but to issue the generation of a complete new snapshot valid at t_{NOW} (*full* refresh).

The second issue which needs an extension addresses the notion of a consistency island comprising all snapshots taken at t_c. As it is easy to imagine, the probability that a huge set of independent snapshots are taken at the same time is very low. To compensate for this drawback, we introduce the notion of an inner consistency band. All snapshots within an inner consistency band are then considered consistent to each other, thus forming the consistency island combined by inner joins. Obviously, the smaller the inner and outer consistency bands are, the more accurately a consistent view of the global database state is achieved.

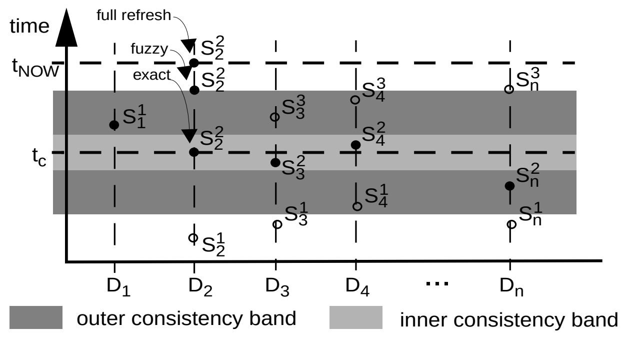


Fig. 6: Example for historic cuts with consistency bands and different refresh strategies

Building a Semi-Consistent Database State for a Given Historic Cut

With the notion of consistency bands, we may now focus on picking the optimal set of snapshots to come up with a global database view which is as consistent as possible. Figure 7 outlines the algorithm *SimpleSnapshotPicking* summarizing the informally mentioned strategies given above. In the example of figure 6, snapshots S_3^2 and S_4^2 are within the inner consistency band and regularly joined together. Snapshots from data sources D_1 and D_n are within the outer consistency band and added to the consistency island with an outer join; snapshot S_2^1 finally needs an update and is rolled forward depending on the maintance mode – exact, fuzzy, full refresh – to t_c, to the upper border of the outer consistency band, or to t_{NOW}. This simple strategy, picking the optimal set of snapshots to produce a semi-consistent database view, will be complicated if the system itself has to determine the optimal historic cut. Different approaches are discussed in the following section.

Moreover, we may add a variation to the before mentioned approach: Instead of specifying t_c directly within the context of a query, we consider a strategy relying on a primary snapshot. The main idea of the primary snapshot approach is based on the primary copy approach used in distributed database system ([ÖzVa91]). The method is based on the fact that exactly one data source D_p is treated in a special way, e.g. holds extremely important data. The time when the most current snapshot of such a data source was taken always determines the historic cut t_c for an incoming query. This primary snapshot approach may be useful beyond preferring a certain data source. It may also be used as a tie-breaker if another strategy does not result in a single solution.

```
Algorithm SimpleSnapshotPicking
Input:    D          // set of data sources $D_i$
                     // each having a set of snapshots $S_i^j$
          $t_c$      // given historic cut
          $\Delta t_o$    // outer consistency band
          strategy   // refresh strategy
Output:   S          // vector of snapshots
begin
    S = [∞,∞,...,∞]
    foreach $D_i$ in D
        // find the closest snapshot to $t_c$
        $S_i^s$ = ∞
        foreach $S_i^j$ in $D_i$
            if ( |time( $S_i^j$ ) - $t_c$| < |time( $S_i^s$ ) - $t_c$| )
                $S_i^s$ = $S_i^j$
            end if
        end foreach
        // check if found snapshot is not within
        // the outer consistency band
        if ( |time( $S_i^s$ ) - $t_c$| > $\Delta t_o/2$ )
            // compute snapshot depending on the strategy
            if ( strategy == exact)
                $S_i^s$ = exact ( $D_i$, $t_c$ )
            elseif ( strategy == fuzzy )
                $S_i^s$ = fuzzy ( $D_i$, $t_o + \Delta t_o/2$ )
            else
                $S_i^s$ = full ( $D_i$, $t_{NOW}$ )
            endif
            // generate snapshot
            generateSnapshot ( $S_i^s$ )
        end if
        S[i] = $S_i^s$
    end foreach
    return S
end
```

Fig. 7: Algorithm *SimpleSnapshotPicking*

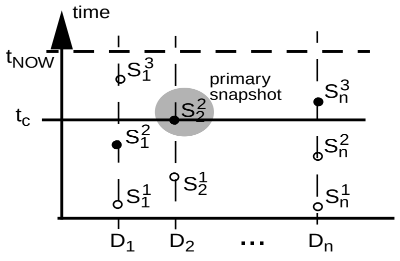


Fig. 8: Historic cut by primary snapshot

Figure 8 holds a sample scenario with D_2 as the primary data source and S_2^2 as the most current and therefore primary snapshot.

4. Generation of Semi-Consistent Views

The algorithms presented in the last section presume a historic cut t_c given either by the user directly or indirectly by the primary snapshot. In this section several algorithms are discussed where the system itself determines the 'optimal' t_c. We start with discussing the optimal mathematical solution. However, since the optimal solution from a mathematical point of view does not always reflect an optimal solution from a user's demand point of view, we propose two algorithmical solutions: a greedy and a cluster based approach.

Quantifying Consistency more Generally

To generally quantify the 'distance' of a snapshot from a global consistent state denoted by a historic cut, we may take advantage of a rich portfolio of different strategies like the number of missing updates, the number of missing updated tuples, value-based distances, or time-related measures. Without any general restrictions, we use the Euclidian norm $\|.\|$ over the distances of all snapshots S_i^j of a single data source D_i and of a historic cut t_c:

$$\|D_i\| = \sqrt{\sum_{j=1}^{k} \left(time\left(S_i^j\right) - t_c\right)^2}$$

The Optimal Mathematical Solution

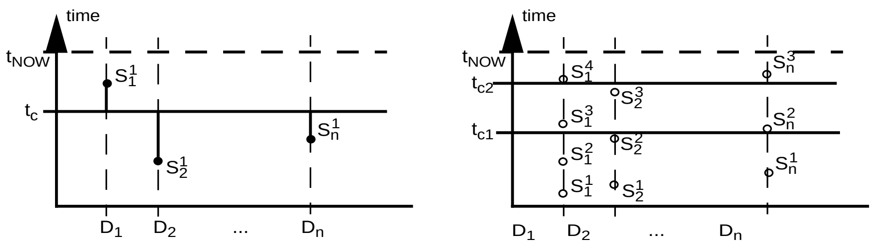
The 'optimal' solution from a mathematical point of view is achieved if the Euclidian distance of a set of snapshots is minimal. With a single snapshot S_i^1 for each data source, we require that the following formula called 'global distance' d_g holds for an optimal historic cut t_c:

$$d_g\left(t_c, \left[S_1^1, S_2^1, \ldots, S_n^1\right]\right) = \sum_{i=1}^{n} \|D_i\|^2 =$$

$$= \sum_{i=1}^{n} \left(time\left(S_i^1\right) - t_c\right)^2 \rightarrow Min$$

Obviously, the solution to this problem is achieved using the first derivation and results in the average over the timestamps of the participating snapshots:

$$t_c = \frac{1}{n} \sum_{i=1}^{n} time\left(S_i^1\right)$$



(a) BG with single snapshot for each data source (b) BG with multiple snapshots for each data source

Fig. 9: Examples for the optimal mathematical solution

This simple mathematical approach may be extended into two directions. At first we may consider the case with k_i snapshots for each data source. Again the average over timestamps of all contributing snapshots yields the solution:

$$t_c = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{k_i} time\left(S_i^j\right)$$

A second extension assigns weights α_i^j to different data sources or different snapshots. More up-to-date snapshots may have a higher weight; unreliable or imprecise data sources may have a lower weight, etc. Extending the solutions above produces the following formulas to compute the optimal historic cut t_c:

•a single snapshot S_i^1 per data source D_i:

$$t_C = \frac{1}{\sum_{i=1}^{n} \alpha_i^1} \cdot \sum_{i=1}^{n} \alpha_i^1 \cdot time\left(S_i^1\right)$$

•multiple snapshots S_i^j per data source D_i:

$$t_c = \frac{1}{\sum_{i=1}^{n} \sum_{j=1}^{k_i} \alpha_i^j} \cdot \sum_{i=1}^{n} \sum_{j=1}^{k_j} \alpha_i^j \cdot time\left(S_i^j\right)$$

Figure 9 (a) illustrates the idea of the formula with a single snapshot for each data source. The historic cut is the line with the minimal distance to all participating snapshots. Figure 9 (b) shows a solution for the general case. The global optimal historic cut is t_{c1}, which determines S_1^3 , S_2^2 and S_n^2 as the snapshots for the query. However, another condition is not considered by this mathematical solution: The historic cut should be as close as possible to t_{NOW}, i.e. t_{c2} in figure 9 (b). However, the computation of t_{c2} cannot be reflected in a single formula, especially when conditions like consistency bands have to be considered. Therefore, we propose two algorithmically working solutions to solve this problem, which is called the 'historic cut picking problem'.

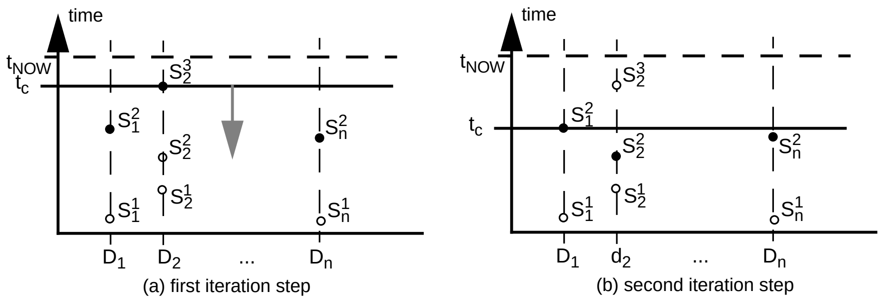
An Iterative Greedy Solution

We propose the following strategy by applying a greedy algorithm to the problem of finding the optimal historic cut: Starting from t_{NOW}, the globally newest snapshot S_i^k is selected, the historic cut t_c is temporarily set to time(S_i^k), and the corresponding set of nearest snapshots S_i^j is computed (following the *Simple-SnapshotPicking* strategy discussed in section 3). Figure 10 (a) shows a scenario with S_2^3 as the most up-to-date snapshot and t_c set to time(S_2^3).

After completing this first iteration, the next step of the algorithm computes a new solution by picking the next newest S_i^k with regard to the current solution t_c. If the global distance of all corresponding snapshots is smaller than the global distance of the current (i.e. first) solution, then the new solution is set to the current solution and the iterative process is continuing. Otherwise the algorithm stops and returns the last (and presumably best) solution. For example, t_c is set to time(S_1^2), because it represents the next newest snapshot (wrt. t_c of figure 10(a)) and the overall distance to the contributing snapshots is less than the global distance with t_c = time(S_2^3). The algorithm *IterativeGreedy* of this iterative approach is presented formally in figure 11.

A Two-Step Clustering Solution

Based on the iterative process, we enrich the current solution to the cut picking problem by applying a two-step clustering algorithm: In the first step for each cluster a local historic (cluster oriented) cut t_{Cu} is computed by applying an already proposed 'global' solution like the mathematical or the iterative approach to each cluster. Finally, we derive a global historic cut t_c by computing the weighted average over the set of all local historic cuts. The challenge of the approach is then to find a proper set of clusters and assign weights to each cluster so that a cluster with important (or most relevant) information over a certain set of data sources exhibits a major impact on determining the global historic cut t_c. Finding clusters may be done in a naive way by putting all snapshots of a single data source into a single cluster; it might be left to the user; it might be based on the idea of partitioning the set of S_i^j so that all snapshots having foreign key relationships or being frequently queried together are put into a single cluster. Gener-



(a) first iteration step (b) second iteration step

Fig. 10: Example of a bubble graph for the greedy algorithm

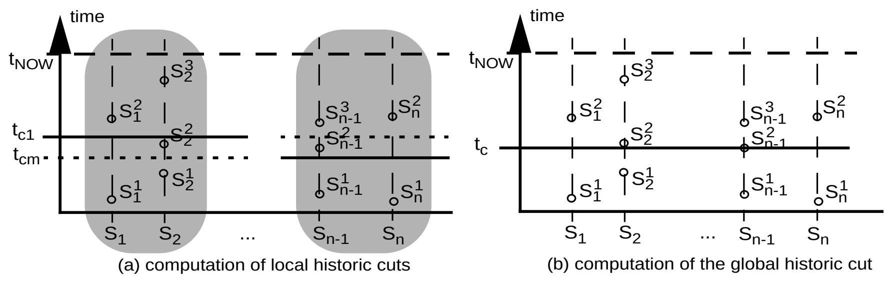

time
t_{NOW}
t_{c1}
t_{cm}
S_1^1 S_1^2 S_2^1 S_2^2 S_2^3
S_{n-1}^1 S_{n-1}^2 S_{n-1}^3 S_n^1 S_n^2
S_1 S_2 ... S_{n-1} S_n
(a) computation of local historic cuts
time
t_{NOW}
t_c
(b) computation of the global historic cut

Fig. 12: Example of a bubble graph for the two-step clustering approach

ally, we think that the partitioning strategy highly depends on the application scenario so that we cannot give an optimal solution here and just rely on an existing partitioning scheme. Figure 12 shows the corresponding examples. Figure 12 (a) illustrates the partitioning schema with the local historic cuts $t_{c1},...,t_{cm}$ which are used to compute the global t_c denoted in figure 12 (b). The algorithm of the approach is given in figure 13 using the function f(D) to compute local historic cuts. [†]

5. Dynamic Algorithm of the Cut Picking Problem

In section 3 an algorithm for finding the closest snapshots is discussed if the historic cut t_c and the inner and outer consistency bands are given. The algorithms for generating semi-consistent views presented in the previous sections are based on partially well-known methods computing a historic cut t_c without any consideration of consistency bands. Extending these methods by explicitly taking the notion of consistency bands into account builds the subject of the first part of this section resulting in a dynamic historic cut picking algorithm. The second part of this section outlines the logical architecture of our prototypical implementation.

```
Algorithm IterativeGreedy
Input: D    // set of data sources D_i each having
            // a set of snapshots S_i^j
Output: t_c // historic cut
        S   // vector of snapshots
begin
    t_new = ∞
    d_new = ∞
    S_new = [∞,∞,...,∞]
    do
        t_c = -∞
        t_old = t_new
        d_old = d_new
        S_old = S_new
        // find next snapshot
        foreach D_i in D
            foreach S_i^j in D_i
                if ( time( S_i^j ) < t_old and time( S_i^j ) > t_c )
                    t_c = time( S_i^j )
                end if
            end foreach
        end foreach
        // compute foreach D_i the closest snapshot to t_new
        foreach D_i in D
            S_i^s = ∞
            foreach S_i^j in D_i
                if ( |time( S_i^j ) - t_c| ≤ |time( S_i^s ) - t_c| )
                    S_i^s = S_i^j
                end if
            end foreach
            t_new = t_c
            S_new[i] = S_i^s
            d_new = d_g( t_new, S_new )
        end foreach
    while ( d_new < d_old )
    return ( t_old, S_old )
end
```

Fig. 11: Algorithm *IterativeGreedy*

Dynamic Cut Picking Algorithm

The most severe situation arises in the context of consistency bands as soon as for one or more data sources all snapshots are outside of the outer consistency band. In this case no snapshot is allowed to participate in the join for the current query and the result given to the user would be undefined. Hence, a refresh operation is introduced to avoid this situation, forcing the system to create a new snapshot of the affected data sources. The refresh mode depends on the update strategy (exact, fuzzy or full – section 3) specified by the user and the type of the data source[‡].

```
Algorithm TwoStepClustering
Input: C    // set of clusters C_i each consisting
            // of a set of data sources D_i
       α_c  // set of weights whereby each weight α_ci
            // corresponds to a cluster C_i
Output:t_c  // historic cut
       S    // vector of snapshots
begin
    // computation of the local historic cuts t_Cu
    // by calling a method f ∈ {SimpleSnaphotPicking, ...
)
    foreach C_um in C
        D = ∅
        foreach D_i in C_um
            D = D ∪ D_i
        end foreach
        t_Cu = f( D )
    end foreach
    // computation of the global historic cut t_c
    t_c = α_C1*t_C1 + ... + α_Cm*t_Cm
    // computation of the set of snapshots
    S = [∞,∞,...,∞]
    foreach C_m in C
        foreach D_i in C_m
            S_i^s = ∞
            foreach S_i^j in D_i
                if ( |time( S_i^j ) - t_c| ≤ |time( S_i^s ) - t_c| )
                    S_i^s = S_i^j
                end if
            end foreach
            S[i] = S_i^s
        end foreach
    end foreach
    return (t_c, S)
```

Fig. 13: Algorithm *TwoStepClustering*

[†] Obviously a clustered configuration may be simulated using the optimal mathematical solution by assigning equal weights to the snapshots within each cluster.

[‡] As already outlined not every data source is able to roll forward to a certain point in time. However, the operation 'full refresh' is always possible resulting in a most up-to-date snapshot.

```
Algorithm DynamicCutPicking
Input:  D         // set of data sources $D_i$
        $\Delta t_o$       // outer consistency band
        strategy  // update strategy
Output: t         // historic cut
        S         // vector of snapshots
begin
    $t_{next} = \infty$ // starting point for searching
    $d_{new} = \infty$                   // magnitude of error
    $S_{new} = [\infty,\infty,...,\infty]$        // vector of snapshots
    $U_{new}$ = [false, false, ..., false]// vector for
                                  // indicating the update
    do
        $t_{old} = t_{new}$
        $d_{old} = d_{new}$
        $S_{old} = S_{new}$
        $U_{old} = U_{new}$
        $U_{new}$ = [false, false, ..., false]
        // find next snapshot
        foreach $D_i$ in D
            foreach $S_i^j$ in $D_i$
                if ( time( $S_i^j$ ) < $t_{next}$ and time( $S_i^j$ ) > $t_{new}$ )
                    $t_{new}$ = time( $S_i^j$ )
                end if
            end foreach
        end foreach
        // compute foreach $D_i$ the closest snapshot to $t_{new}$
        foreach $D_i$ in D
            $S_i^s = \infty$
            foreach $S_i^j$ in $D_i$
                if ( |time( $S_i^j$ ) - $t_{new}$| ≤ |time( $S_i^s$ ) - $t_{new}$| )
                    $S_i^s = S_i^j$
                end if
            end foreach
            $S_{new}[i] = S_i^s$
        end foreach
(continuation on the right side)
        // check if all $S_i^j$ are inside
        // the outer consistency band
        validSolution = true;
        foreach $S_i^j$ in $S_{new}$
            if ( |time( $S_i^s$ ) - $t_{new}$| > $\Delta t_o/2$ )
                // compute the snapshot S depending
                // on the refresh strategy
                if ( strategy == exact)
                    S = exact ( $D_i$, $t_{new}$ )
                else if ( strategy == fuzzy )
                    S = fuzzy ( $D_i$, $t_{new} + \Delta t_o/2$ )
                else
                    S = full ( $D_i$, $t_{new}$ )
                end if
                // store S in the snapshot vector
                // if it is within the outer consistency band
                if ( $S \le t_{new} + \Delta t_o/2$ and $S \ge t_{new} - \Delta t_o/2$ )
                    $S_{new}[i]$ = S
                    $U_{new}[i]$ = true
                else
                    validSolution = false
                end if
            end if
        end foreach
        // compute magnitude of error and set other variables
        $t_{next} = t_{new}$
        if ( validSolution = true )
            $d_{new} = d_g$ ( $t_{new}$, $S_{new}$ )
        else
            $t_{new} = t_{old}$
            $d_{new} = d_{old}$
            $S_{new} = S_{old}$
            continue;
        end if
    while ($d_{new} < d_{old}$)
    // generate snapshots
    foreach $U_i$ in $U_{old}$
        if ( $U_i$ == true )
            generateSnapshot ( $S_{old}[i]$ )
        end if
    end foreach
    return ($t_{old}$, $S_{old}$)
end
```

Fig. 14: Algorithm *DynamicCutPicking*

The main idea of the dynamic cut picking algorithm (figure 14) is based on the iterative greedy approach (section 4, figure 11). The algorithm additionally checks for every data source D_i if the snapshot S_i^j being closest to the current historic cut is within the outer consistency band[**]. If not the algorithm assumes the existence of a new snapshot. If this potentially generated snapshot is outside the outer consistency band in case of a required full refresh, the corresponding historic cut represents an invalid solution. As in the *IterativeGreedy*-algorithm the algorithm terminates if the global distance is not further improved by computing a new historic cut and the current solution is accepted as the best one. Since this solution might involve potentially new snapshots, these have to be generated before returning the historic cut t_c to the user. Running this algorithm in the 'strong' mode, it might be possible that no valid solution is returned to the user (especially in combination with a full refresh). Therefore, a 'weak' solution is signalled to the user, if a generated or existing snapshot is outside the outer consistency band and the snapshot being closest to the historic cut is added to the solution.

Moreover, the dynamic cut picking algorithm may be extended to a cluster algorithm. As the *TwoStepClustering*-algorithm (figure 13) builds a general framework, the *DynmicCutPicking*-algorithm can be used inside the cluster algorithm to compute a cluster oriented local historic cut.

[**] It is assumed that the historic cut forms the center of the outer consistency band. Without giving any further details, we may notate that non-uniform allocations are possible.

Logical Architecture

The proposed algorithms are currently under implementation. Figure 15 sketches the logical architecture of the system. Snapshots of each data source D_i are accessed by wrappers and stored inside a relational database system. Meta information regarding the snapshots, e.g. the corresponding data source, the number of snapshots per data source, the timestamp when having taken the snapshots, etc., are stored inside the repository. This meta information is then used by the query snapshot rewriter to transparently transform a query Q_R issued by a user based on table names of the data sources into a sequence of update statements and queries Q_S targeting different snapshots.

To specify the inner and outer consistency bands as well as the refresh strategy at the user level, we extend the USING clause of a SELECT statement. The acronyms *icb* and *ocb* denote the parameter (data type: time interval) for the inner and outer consistency bands. The keyword *refresh* is used to determine the refresh strategy (*refresh-mode*) which is one out of exact, fuzzy or full. Finally the strategy to solve the cut finding problem has to be specified in the parameter *cfp-strategy*:

```
USING  icb = TIMEINTERVAL AND
       ocb = TIMEINTERVAL AND
       refresh = REFRESH-MODE AND
       strategy = CFP-STRATEGY
```

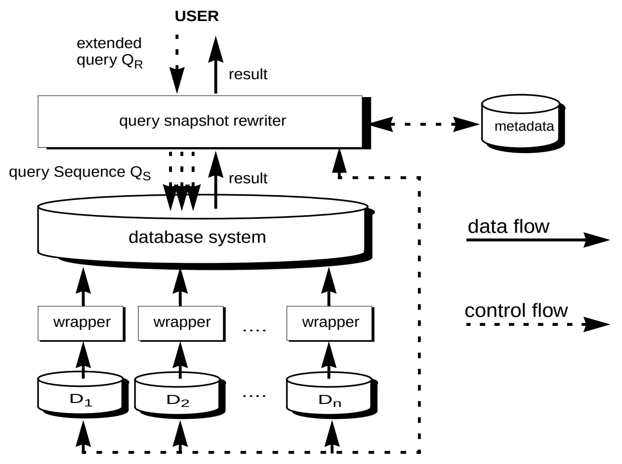


Fig. 15: Semi-consistent view generation using a query snapshot rewriter

Using this information, a given SQL query is rewritten. For example, consider the three data sources D_1 holding table $R_1(A,B)$, data source D_2 with $R_2(B,C)$, and D_3 with $R_3(C,D)$ and the following natural join query over tables of the three data sources:

```
SELECT *
FROM   R1, R2, R3
WHERE  R1.B = R2.B AND
       R2.C = R3.C
USING  icb = 3s AND
       ocb = 5s AND
       refresh = fuzzy AND
       cfp_strategy = (primary_snapshot, D2)
```

In figure 16 the situation of existing snapshots is visualized . Since D_2 is required to act as the primary snapshot, t_c is set to the generation time of the newest snapshot S_2^2 of D_2. Obviously, S_1^2 is selected as the proper snapshot for data source D_1, because it exhibits the minimal local distance to t_c. Finally the snapshot S_3^1 is the only snapshot for D_3. Since it is outside the outer consistency band, a fuzzy refresh is executed and S_3^2 is generated. For this example, the original user query is resolved into two statements, a CREATE TABLE statement for the generation of the snapshot and a SELECT statement for the query itself, which are issued against the database system:

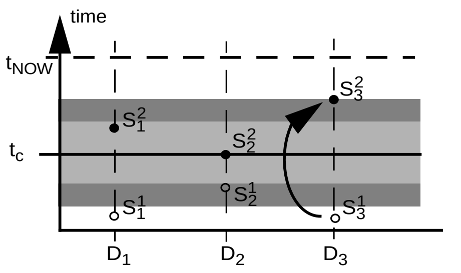


Fig. 16: Bubble graph of the rewrite example

```
CREATE TABLE S32 AS
    (SELECT *
     FROM D3
     WHERE TIMESTAMP = generationTime(S22) +
ocb);
SELECT *
```

6. Summary and Future Work

Retrieving information from data sources is one of the most important challenges in information processing. While most of the ‘classical’ approaches have still the ‘perfect world’ model in mind, we argue that a user might be satisfied with an answer reflecting not the most current and most accurate database view. Therefore, we introduce an information system architecture based on the storage of multiple snapshots together with a consistency framework comprised of inner and outer consistency bands for user queries. Based on this framework we outline different strategies to come up with a semi-consistent view over the set of participating data sources. Moreover, the logical architecture of our prototypical implementation of a query rewrite engine, managing, i.e. creating and dropping snapshots and transforming a query to the selected set of snapshots is sketched at the end of the paper.

Regarding future work we intend to extend the distance metric. As we only consider the actuality problem in this paper, our effort is currently in integrating the aspect of data changes into the metric.

To conclude, we think that our approach is suitable as an add-on to existing middleware systems and may considered as a contribution to query autonomus data sources.

Acknowledgement

We would like to express our most sincerely ‘Thank You’ to Wolfgang Hümmer for the fruitful and inspiring discussion during the preparation of this paper.

References

AdLi80 Adiba, M.E.; Lindsay, B.G.: Database Snapshots. In: *Proceedings of the 6th International Conference on Very Large Data Bases (VLDB'80, Montreal, Canada, Oct 1 - 3)*, 1980, pp. 86-91

BeGo83 Bernstein, P.; Goodman, N.: Multiversion Concurrency Control - Theory and Algorithms. In: *ACM Transactions in Database Systems 8(4)*, 1983, pp. 465-483

BrGS92 Breitbart, Y.; Garcia-Molina, H.; Silberschatz, A.: Overview of Multidatabase Transaction Management. In: *VLDB Journal 1(2)1992*, pp. 181-293

GHRU97 Gupta, H.; Harinarayan, V.; Rajaraman, A.; Ullman J.: Index Selection for OLAP. In: *Proceedings of the 13th International Conference on Data Engineering (ICDE'97, Birminghan, 7.-11. April)*, 1997, pp. 208-219

GrRe93 Gray, J.; Reuter, A.: *Transaction Processing - Concepts and Techniques*. Morgan Kaufmann Publishers, San Mateo, 1993

GuMu99 Gupta, A.; Mumick, I.S.: *Materialized Views : Techniques, Implementations, and Applications*. MIT Press, 1999

IBM01a N.N.: *DataJoiner*, IBM Corp., 2001 (electronic version: http://www-4.ibm.com/software/data/datajoiner/)

IBM01b N.N.: *The Garlic Project*, IBM Corp., 2001 (electronic version: http://www.almaden.ibm.com/cs/garlic/ homepage.html)

IBM01c N.N.: *The TSIMMIS Project*, 2001 (http://www-db.stanford.edu/tsimmis/tsimmis.html)

Inmo96 Inmon, W.H.: *Building the Data Warehouse*, 2nd edition. New York, Chichester, Brisbane, Toronto, Singapur: John Wiley & Sons, Inc., 1996

IsBV98 Isakowitz, T.; Bieber, M.; Vitali, F.: Web Information Systems - Introduction. In: *Communication oif the ACM (CACM) 41(1998)7*, pp.78-80

JCE+94 Jensen, C.S.; Clifford, J.; Elmasri, R.; Gadia, S.K.; Hayes, P.J.; Jajodia, S.: A Consensus Glossary of Temporal Database Concepts. In: *SIGMOD Record 23(1)1994*, pp. 52-64

Kimb96 Kimball, R.: *The Data Warehouse Toolkit*, 2nd edition. New York, Chichester, Brisbane, Toronto, Singapur: John Wiley & Sons, Inc., 1996

LaGa96 Labio, W.; Garcia-Molina, H.: Efficient Snapshot Differential Algorithms for Data Warehousing. In: *Proceedings of the 22th International Conference on Very Large Data Bases (VLDB'96, Bombay, Sept 3 - 6)*, 1996, pp. 63-74

LHM+86 Lindsay, B.G.; Haas, L.M.; Mohan, C.; Pirahesh, H.; Wilms, P.F.: A Snapshot Differential Refresh Algorithm. In: *Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data (SIGMOD'86, Washington, D.C., USA, May 28-30)*, 1986, pp. 53-60

Lint99 Linthicum, D.S.: *Enterprise Application Integration*, Addison-Wesley, 1999

LuZP01 Lu, B.; Zou, Q.; Perrizo, W.: A dual copy method for transaction separation with multiversion control for read-only transactions. In: *Proceedings of the 2001 ACM Symposium on Applied Computing (SAC'01, Las Vegas, Nevada, USA, March 11-14)*, 2001, pp. 290-294.

RoSc97 Roth, M.T.; Schwarz, P.M.: Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources. In: *Proceedings of 23rd International Conference on Very Large Data Bases (VLDB'97, Athens, Greece, August 25-29)*, 1997, pp. 266-275

SBCL00 Salem, K.; Beyer, K. S.; Cochrane, R.; Lindsay, B. G.: How To Roll a Join: Asynchronous Incremental View Maintenance. In: *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD'00, Dallas, Texas, USA, May 16 - 18),* 2000, pp. 129-140

ShLa90 Sheth, A.P.; Larson, J.A.: Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. In: *ACM Computing Surveys 22(3)1990*, pp. 183-236

TCG+93 Tansel, A.; Clifford, J.; Gadia, S.; Jajodia, S.; Segev, A.; Snodgrass, R.: *Temporal Databases*. Benjamin/Cummings Publisching, Redwood City, 1993

WeVo01 Weikum, G.; Vossen, G.: *Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control*, Morgan Kaufmann Publishers, 2001

Wied92 Wiederhold, G.: Mediators in the architecture of future information systems. In: *IEEE Computer 25(3)1992*, pp. 38-49

ZCL+00 Markos Zaharioudakis, Roberta Cochrane, George Lapis, Hamid Pirahesh, Monica Urata: Answering Complex SQL Queries Using Automatic Summary Tables. In: *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD'00, Dallas, Texas, USA, May 16 - 18)*, 2000, pp. 105-116

ÖzVa91 Özsu, M.; Valduriez, P.: *Principles of Distributed Database Systems*. Prentice-Hall, Englewoods Cliffs (New Jersey, USA), 1991