Exploiting Versions for Online Warehouse Maintenance in MOLAP Servers

Heum-Geun Kang and Chin-Wan Chung
KAIST
VLDB 2002
Organizations

• Introduction
• Multi-Dimensional Arrays (MDAs) for MOLAP
• Multi-Version Concurrency Control for Data Warehouses (MVCC-DW)
• Experiments
• Conclusion
Introduction

• Data warehouses
  – Enable users to make better and fast decisions
  – Collect information from several data sources
  – Support online analytical processing

• OLAP
  – Multi-dimensional OLAP (MOLAP)
  – Relational OLAP (ROLAP)
Introduction

- **Query Transaction**
  - Sequence of interactive queries
  - Queries tend to be complex and involve large volumes of data

- **Maintenance transaction**
  - Gather changes to the source data and propagate the changes to the warehouse data
  - Executed periodically

- **The differences between OLAP and OLTP**
  - Transaction execution time
  - The number of update transactions (maintenance transactions)
  - Volume of data to be accessed
Introduction

- It has been known that the CC mechanisms for OLTP systems are not adequate for OLAP systems.
- A naive method:
  - Not to run queries during the maintenance time.
  - As corporations become globalized, the OLAP systems should be able to respond to the queries submitted by users in multiple time zones.
Contribution

• Propose MVCC-DW suited for data warehouses managed by MOLAP servers
• Features of the MVCC-DW
  – Non Blocking
  – No Lock
• Prove the correctness of MVCC-DW
• Implement the MVCC-DW mechanism
• Show the MVCC-DW mechanism works efficiently
MDAs for MOLAP

- A set of B-trees
  - Map dimension values to array index values
  - One for each dimension
- A multi-dimensional index
  - Maps a sequence of array index values to a chunk
  - A chunk is a small multi-dimensional array
- A chunked file
  - Stores a set of chunks rather than a large array
MVCC-DW

• Motivation and Idea
  – Locking mechanisms result in a high blocking rate
  – Optimistic concurrency control mechanisms can have a high abort rate of long transactions
  – Our basic idea is to use a version mechanism
  – A chunk instead of a cell is used as the unit of version control
  – We devise a new access method which supports the versioning concept
MVCC-DW(Revision)

- A revision is a snapshot of the data warehouse
- State
  - Active: being changed
    - There is at most one at a time
  - Frozen: is not changed anymore
- Current: the most recently frozen revision
- Oldest: the least recently frozen revision
- Every revision is assigned a revision number
MVCC-DW (Revisions and Transactions)

- A transaction uses only one revision throughout the lifetime

<table>
<thead>
<tr>
<th>Who</th>
<th>When</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>maintenance transaction</td>
<td>begin</td>
<td>create an active revision</td>
</tr>
<tr>
<td>maintenance transaction</td>
<td>end</td>
<td>freeze the active revision</td>
</tr>
<tr>
<td>query transaction</td>
<td>begin</td>
<td>open the current revision</td>
</tr>
<tr>
<td>query transaction</td>
<td>end</td>
<td>close the opened revision</td>
</tr>
</tbody>
</table>
Multi-Revision (MRV)-tree: manages information about revisions

A set of B+ trees

Chunked File

< The architecture of multi-versioned 2-dimensional array >
MVCC-DW(Arch. and Data Structures)

- MRV-tree
  - Consists of a root node and a sequence of ReVision$^n$(RV$^n$)-trees
  - Adjacent RV$^n$-trees share nodes if the nodes are not changed

< An example of the MRV-tree >
MVCC-DW(Arch. and Data Structures)

- **RV$_n$-tree**
  - Manages the chunks contained in a revision
  - Maps a sequence of index values to a chunk
  - Each node has a revision number of a revision that was active at node creation time

**Directory Node**

<table>
<thead>
<tr>
<th>rNum</th>
<th>C$_1$,MBR$_1$</th>
<th>C$_2$,MBR$_2$</th>
<th>C$_3$,MBR$_3$</th>
<th>...</th>
</tr>
</thead>
</table>

(rNum : revision number, C$_i$ : address of child node i)

**Leaf Node**

<table>
<thead>
<tr>
<th>rNum</th>
<th>O$_1$,MBR$_1$</th>
<th>O$_2$,MBR$_2$</th>
<th>O$_3$,MBR$_3$</th>
<th>...</th>
</tr>
</thead>
</table>

(rNum : revision number, O$_i$ : id of object storing chunk i)
MVCC-DW(CreateRevision)

- Creates a revision and sets the revision to active
MVCC-DW(FreezeRevision)

- Closes the active revision and makes the revision frozen so that query transactions can retrieve the updated data in the revision
**MVCC-DW (Garbage Collection)**

- The **oldest revision** that is not the current revision can be released.
- Since adjacent revisions share some nodes and chunks, a care is required.
MVCC-DW(Correctness)

- Lemma 1: Let $T_i$ use the revision $n$ and $T_j$ use the revision $m$ where $n < m$. There is no edge from $T_j$ to $T_i$ in a serialization graph.
- Lemma 2: There is no cycle between transactions using the same revision.
- Theorem: An SG(H) for a history H produced by MVCC-DW is acyclic.
**MVCC-DW (Clustering of cells)**

- The number of chunks to be versioned has an impact on the performance of MVCC-DW.
- The whole chunk is versioned even if a cell in the chunk is updated.
- It is desirable that all cells in a chunk have the same time dimension value.
Experiments

• Data set: APB Benchmark
  – Dimensions: customer, product, channel, time
  – Composed of historical data and incremental data
  – The number of valid cells in the data cube constructed from the historical data: 21,000,000

• Prototype
  – Built by modifying the Shore storage manager. Particularly, R*-tree
Experiments

<table>
<thead>
<tr>
<th></th>
<th>Customer</th>
<th>Product</th>
<th>Channel</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size A</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>Size B</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>Size C</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>1</td>
</tr>
<tr>
<td>Size D</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>1</td>
</tr>
<tr>
<td>Size E</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>1</td>
</tr>
</tbody>
</table>

< Chunk Sizes >
### Experimental Results

#### Table 3: Status of the RV0-tree and the chunked file after the historical data is loaded

<table>
<thead>
<tr>
<th>Chunk Size</th>
<th>RV0-tree Nodes</th>
<th>Num. of chunks</th>
<th>Valid cells in chunks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>103</td>
<td>13,522</td>
<td>1,503,000</td>
</tr>
<tr>
<td>B</td>
<td>40</td>
<td>5,097</td>
<td>600,525</td>
</tr>
<tr>
<td>C</td>
<td>17</td>
<td>2,381</td>
<td>281,100</td>
</tr>
<tr>
<td>D</td>
<td>9</td>
<td>1,318</td>
<td>156,250</td>
</tr>
<tr>
<td>E</td>
<td>7</td>
<td>862</td>
<td>100,000</td>
</tr>
</tbody>
</table>

#### Table 4: Status of the RV1-tree and the chunked file after the incremental data is loaded

<table>
<thead>
<tr>
<th>Chunk Size</th>
<th>Num. of RV1-tree nodes</th>
<th>Num. of Chunks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>revision 0</td>
<td>revision 1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>A</td>
<td>7,419</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>2,944</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1,477</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>812</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>550</td>
<td>1</td>
</tr>
</tbody>
</table>
Conclusion

- Conventional concurrency control mechanisms are not adequate for a data warehouse environment
- Proposed a multi-version concurrency control mechanism, MVCC-DW, that exploits versions for online data warehouse maintenance in MOLAP servers
- Demonstrated the efficiency of MVCC-DW
  - The number of versioned chunks is small