On Parallel Processing of Aggregate and Scalar Functions in Object-Relational DBMS

Michael Jaedicke, Bernhard Mitschang

ACM SIGMOD 1998, 06/04/1998

Overview:

- Introduction to user-defined functions
- Parallel processing of UDFs
  - a parallel processing scheme for user-defined aggregate functions
  - a classification of user-defined data partitioning functions
  - parallel sorting as preprocessing step for aggregate functions
User-Defined Functions (UDFs) in ORDBMS

- User-defined **scalar functions** (UDSFs):
  - $f$: data items $\rightarrow$ data item
  - examples: concat, +, ceiling, hex, rand, dayofyear, ...

- User-defined **aggregate functions** (UDA Fs):
  - $f$: set of data items $\rightarrow$ data item
  - examples: avg, sum, count, max, min, variance, ...

- Not covered here:
  - user-defined table functions
  - user-defined support functions (for internal purposes)
Registration of UDFs

- Registration: define a new UDF and provide metadata for it

- Example (DB2 UDB):

```sql
CREATE FUNCTION distance (point, point)
RETURNS double
EXTERNAL NAME 'point!distance'
LANGUAGE C
PARAMETER STYLE DB2SQL
NOT VARIANT
NOT FENCED
NOT NULL CALL
NO SQL
NO EXTERNAL ACTION
NO SCRATCHPAD
NO FINAL CALL;
```

- **no input context**
- **no external context**
Sequential Processing of UDAFs

- UDAFs processed by means of iterator concept (one tuple at a time)
  - Aggregation needs temporary storage for intermediate results (of sum, count, avg, ...)
- Example (Illustra):
  Initialize and terminate aggregation by means of functions that are provided with the registration: Init, Iter and Final

```
pointer = Init()

Iter(pointer, value)

value = Final(pointer)
```

- All functions that compute aggregate functions have an input context
Parallel Processing of Built-in Functions

- Goal: partitioned parallelism
- Data partitioning and parallel processing schemes

1-step scheme for scalar functions

2-step scheme for aggregate functions

Fixed, built-in parallel processing schemes
2-Step Parallel Aggregation for UDAFs

- Goal: enable parallel processing of user-defined aggregate functions
- Idea: make traditional 2-step processing scheme available for UDAFs
- Difference between built-in and user-defined aggregate functions:
  Developer has to define local and global aggregate functions

☞ Extend the CREATE AGGREGATE statement:

```
CREATE AGGREGATE <function-name> 
( 
 LOCAL <Init, Iter, and Final function definition>
 GLOBAL <Init, Iter, and Final function definition>
 )
```

☞ Straightforward extension of current ORDBMS
Extension of the 2-Step Processing Scheme

built-in aggregate functions

user-defined aggregate functions

Local Agg.  Local Agg.  Local Agg.

Global Aggregation

Local UDAF  Local UDAF  Local UDAF

Global UDAF

MERGE

DATA PARTITIONING

data
Data Partitioning: A Limit of the 2-Step Scheme for UDAFs

Example: compute the most frequent value of a set

Approach: implement Most_Frequent with the 2-step processing scheme
- local aggregation: compute number of the most frequent value for each partition
- global aggregation: select the value with the highest local frequency

Problem: if the same value occurs in several partitions, the result is not correct

For some UDFs it is not correct to use an arbitrary partitioning of the data

Developer has to tell the DBMS, how the data partitioning has to be done for a given UDF
Data Partitioning and UDFs

- Goal: extensibility of parallel processing schemes with respect to data partitioning
- Data partitioning can be described by means of partitioning functions
- Idea: allow user-defined partitioning functions

- First approach:
  developer specifies only a single specific data partitioning function for each UDF

- Problem: if several UDFs have to be computed data repartitioning is necessary

  ➤ not the best solution
Classes of Data Partitioning Functions

- Goal: avoid data repartitioning
- Idea: classification of partitioning functions; developer specifies a class of applicable partitioning functions

- Classes of data partitioning functions:
  - **ANY**: round robin, random
  - **EQUAL**: hash
  - **RANGE**: range partitioning

  ➞ ANY ⊇ EQUAL ⊇ RANGE

- If no class can be applied for a UDF, try
  - a single, specific user-defined data partitioning function for example a spatial data partitioning function
Example: Registration of the Function Most_Frequent

- Registration of the (local) Iter function with partitioning class EQUAL for the UDAF Most_Frequent:

```sql
CREATE FUNCTION Most_Frequent_ITER_LOCAL(POINTER, INTEGER)
RETURNS POINTER
EXTERNAL NAME 'libfuncts!mf_iter_local'
ALLOW PARALLEL WITH PARTITIONING CLASS EQUAL $2
LANGUAGE C ...;
```
Avoiding Data Repartitioning

Example: use partitioning classes to avoid data repartitioning

SELECT Count(*), Most_Frequent(Job)
FROM Staff

Count(*): \text{ANY}
Most_Frequent: \text{EQUAL}

Query optimizer:

\text{ANY} \cap \text{EQUAL} = \text{EQUAL}
Partitionable UDFs

- Goal: describe which UDFs can be processed in parallel

- A UDSF is *partitionable for class C*, iff the function
  - can be processed in parallel using any partitioning function of class C

- A UDAF is *partitionable for class C*, iff the function
  - can be processed using the 2-step processing scheme (local and global aggregation) and
  - the local aggregate function can be processed in parallel using any partitioning function of class C
Parallel processing schemes can be made extensible by means of user-defined partitioning functions.
Limited Applicability of the 2-Step Scheme

How to compute the median of a set in parallel with the 2-step scheme?

```
SELECT Median(P.Age, COUNT(*))
FROM Pers AS P
```

New approach based on parallel sorting:

- sort the input set in parallel
- scan the sorted input until the position of the median is reached
- return the median

No suitable local aggregate function ?!?
Parallel Sorting as a Preprocessing Step

- **Goal:** support limited “parallel” processing, if the 2-step scheme fails
- **Idea:** allow UDFs that operate on a sorted input; DBMS can sort in parallel as a preprocessing step

An aggregate function \( f \) that requires a sorted input can be evaluated using the following scheme given an input set \( S \):

- sort the input set \( S \); this can be done in parallel
- compute \( f \) without parallelism on the sorted input

- Registration of the local Iter function for the UDAF **Median**:

```sql
CREATE FUNCTION MEDIAN_ITER_LOCAL(POINTER, INTEGER)
RETURNS POINTER
EXTERNAL NAME 'libfuncs!median_iter_local'
ORDER BY $2
LANGUAGE C . . .;
```
Related Work

☐ Goal: efficient computation of Data Cubes (Jim Gray et al)

☐ 3 disjoint classes of aggregate functions:

☞ Distributive aggregate functions:
  sub-aggregates can be computed on arbitrary sub-sets with the aggregate function itself
  ➤ partitionable for class ANY

☞ Algebraic aggregate functions:
  sub-aggregates with fixed size can be computed on arbitrary sub-sets
  ➤ partitionable for class ANY

☞ Holistic aggregate functions
  sub-aggregates with fixed size cannot be computed on arbitrary sub-sets
  ➤ partitionable for some data partitioning function (not ANY)
  or
  not partitionable, but parallel sorting might help
## Summary

- **User-defined functions: context and parallel processing**

<table>
<thead>
<tr>
<th>Context</th>
<th>Scalar Functions</th>
<th>Aggregate Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>no context</td>
<td>partitionable for class ANY</td>
<td>-</td>
</tr>
<tr>
<td>input context</td>
<td>partitionable for some class</td>
<td>partitionable for some class with local and global aggregation</td>
</tr>
<tr>
<td></td>
<td>not partitionable</td>
<td>parallel sorting</td>
</tr>
<tr>
<td>external context</td>
<td>not treated here</td>
<td>not partitionable</td>
</tr>
</tbody>
</table>