University of ZurichUZH

Department of Informatics

University of Zürich
Department of Informatics
Binzmühlestr. 14
CH-8050 Zürich
Phone. +41 44 635 43 11
Fax +41 44 635 68 09
www.ifi.uzh.ch/dbtg

UZH, Dept. of Informatics, Binzmühlestr. 14, CH-8050 Zürich

Urs Vögeli

Prof. Dr. Michael Böhlen
Professor
Phone +41 44 635 43 33
Fax +41 44 635 68 09
boehlen@ifi.uzh.ch

Zürich, October 14, 2013

Facharbeit in Informatik
Datenbanktechnologie

Topic: The aggregate & buffer effect in the CC-Join operator

Overview:

Given an outer and an inner relation, the CC-Join is an operator computing the equijoin according to a common identifier, plus the nearest neighbor join according to the time for all outer tuples not having an equijoin match. Since the time is not an identifier (i.e. many tuples could be present in the inner relation with the same timestamp), all nearest neighbors of a given outer tuples are aggregated.

The goal of this Facharbeit is to design, define, implement, and evaluate an extension of the CC-Join operator offering the possibility to avoid the aggregation of the multiple join matches.

Detailed description:

Given an outer relation $\mathbf{r}$ $[E,G,T\,]$ and in inner relation s $[E,G,T,K,M\,]$, the CC-Join operator is defined as following:

$$\mathbf{r} \underset{\mathrm{EQ(E)}}{\overset{\mathrm{NN(T)}}{\bowtie}} \mathbf{s}_{k} = {}_{\mathbf{r}.*}\vartheta_{f(\mathbf{s}_{k}.M)}\big(\mathbf{r} \bowtie_{\mathbf{E,G}} \mathbf{s}_{k} \cup \big((\mathbf{r} \rhd_{\mathbf{E,G}} \mathbf{s}_{k}) \overset{\mathrm{NN(T)}}{\bowtie}_{\mathbf{G}} \mathbf{s}_{k}\big)\big)$$

where $\mathrm{s}_{k} = \sigma_{K=k}(\mathrm{s})$ and $\bowtie_{\mathbf{E,G}}$ is the equijoin operator on E, $\rhd_{\mathbf{E,G}}$ the antijoin on E, $\overset{\mathrm{NN(T)}}{\bowtie}_{\mathbf{G}}$ the nearest neighbor join on T and ϑ the aggregation operator. Attribute G is a grouping attribute used for specifying that the join matches of a given $\mathbf{r}$ tuple, must be found among the s_{k} tuples sharing the same G value; T is a temporal attribute.

In the following example, we define a partion of $\mathbf{r}$ as $\mathbf{r}_g = \sigma_{G=g}(\mathbf{r})$, a partition of s as $\mathrm{s}_{g,k} = \rho_{M/k}(\sigma_{G=g \wedge K=k}(\mathrm{s}))$ and a set of partitions of s as $\mathrm{s}_{k} = \sigma_{K=k}(\mathrm{s})$ and we refer to a tuple $r \in \mathbf{r}$ as a *query point.*

Consider the following expression that we use in the Swiss Feed Data Warehouse to retrieve the Crude Protein cp and the Organic Matter om values of a set of query points $\mathbf{r}$: the cp values are stored in $\mathbf{s}_{cp}$ while the om values are stored in $\mathbf{s}_{om}$.

$$\mathbf{Q} = \Pi_{\mathbf{E},\mathbf{G},T,om,cp}\left(\mathbf{r} \underset{\mathrm{EQ(E)}}{\overset{\mathrm{NN(T)}}{\bowtie}} \mathbf{s}_{cp} \underset{\mathrm{EQ(E)}}{\overset{\mathrm{NN(T)}}{\bowtie}} \mathbf{s}_{om}\right)$$

For a given query point $r \in \mathbf{r}$, if a tuple $s \in \mathbf{s}_{cp}$ (or $\mathbf{s}_{om}$) does not exists with $s.E = r.E$ and same G, we consider as its cp (or om) value the temporal closest tuple in $\mathbf{s}_{cp}$ (or $\mathbf{s}_{om}$) with same G.

r

		E	G	T
$\mathbf{r}_{hay}$	r_1	#111	Hay	2011-05-21
	r_2	#222	Hay	2011-06-21
	r_3	#333	Hay	2011-07-21
$\mathbf{r}_{pea}$	r_4	#444	Pea	2011-07-21

s

			E	G	K	A	T	R	M
$\mathbf{s}_{cp}$	$\mathbf{s}_{hay,cp}$	s_1	#111	Hay	CP	1030	2011-05-21	0.9	140
		s_2	#221	Hay	CP	1000	2011-06-20	0.9	107
		s_3	#223	Hay	CP	1280	2011-06-22	0.9	109
		s_4	#330	Hay	CP	1400	2011-07-19	0.9	94
	$\mathbf{s}_{pea,cp}$	s_5	456	Pea	CP	1000	2011-01-02	0.8	1.06
$\mathbf{s}_{om}$	$\mathbf{s}_{hay,om}$	s_6	#111	Hay	OM	1030	2011-05-21	0.9	885
		s_7	#224	Hay	OM	940	2011-06-23	0.9	890
		s_8	#225	Hay	OM	1080	2011-06-24	0.9	900
		s_9	#333	Hay	OM	1200	2011-07-21	0.9	910
	$\mathbf{s}_{pea,om}$	s_{10}	456	Pea	OM	1000	2011-01-02	0.8	950
	$\mathbf{s}_{pea,om}$	s_{11}	456	Pea	OM	1000	2011-01-02	0.8	946

q

		E	G	T	CP	OM
$\mathbf{r}_{hay}$	q_1	#111	Hay	2011-05-21	140	885
	q_2	#222	Hay	2011-06-21	108	900
	q_3	#333	Hay	2011-07-21	94	910
$\mathbf{r}_{pea}$	q_4	#444	Pea	2011-07-21	106	948

Table $\mathbf{Q}$ is obtained applying the definition of the CC-Join operator between the query points $\mathbf{r}$ and the partition sets $\mathbf{s}_{cp}$ and $\mathbf{s}_{om}$. Since possible multiple join matches are aggregated, each query point r can result at most into one result tuple. For example, for query point r_2 no equijoin match on E exists but two nearest neighbors on T exist in $\mathbf{s}_{cp}$: tuples s_2 and s_3 are both at 1 day distance from r_2. The operator joins therefore r_2 with the average value among $s_2.M$ and $s_3.M$

The goal of this project is to build an extension of the CC-Join able to provide table $\mathbf{Q}'$:

q'

		E	G	T	CP	OM
$\mathbf{r}_{hay}$	q_1	#111	Hay	2011-05-21	140	885
	q_2	#222	Hay	2011-06-21	107	900
	q_3	#222	Hay	2011-06-21	109	900
	q_4	#333	Hay	2011-07-21	94	910
$\mathbf{r}_{pea}$	q_5	#444	Pea	2011-07-21	106	950
	q_6	#444	Pea	2011-07-21	106	946

Table $\mathbf{Q}$' is obtained extending the definition of the CC-Join with the possibility that multiple join matches are not aggregated but result in multiple output tuples. Remembering that for query point r_2 two tuples (s_2 and s_3) exist in $\mathbf{s}_{cp}$ at the same minimum distance, $\mathbf{Q}$' contains a tuple q_2 resulting by the join between r_2 and s_2 and a tuple q_3 resulting by the join between r_2 and s_3.

The Algorithm will be implemented in C++ extending the engine of the DBMS of PostgreSQL and should be called at query level as follows:

```
SELECT E, G, T, CP, M AS OM
FROM (SELECT E, G, T, M AS CP
      FROM  r CCJOIN s_cp EQUAL ON E NN BY G USING T [AGG M]
      ) r CCJOIN s_om EQUAL ON E NN BY G USING T [AGG M]
```

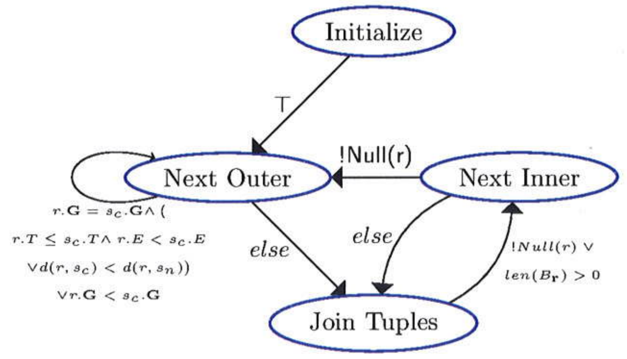
where the optionality of the AGG clause, establish if multiple join matches have to be aggregated or not.

Implementation:

The CC-Join has been integrated in the kernel of PostgreSQL as a sort merge procedure that takes advantage from the fact that multiple join matches are aggregated and avoids to backtrack (re-fatching already scanned rows) any of the two input relations. The approach uses two additional structures:

- one buffer $B_\mathbf{r}$ storing a scanned outer tuple missing an equijoin match, until all its nearest neighbors are fetched
- one aggregation set storing the aggregation of the nearest neighbors of the current outer tuple since it may be used as well as join match of the next outer tuple.

The approach is modeled by a finite state machine composed by 4 main states and available in the file *nodeCCJoin.c* of the implementation that will be given to the student.


Initialize
T
Next Outer
!Null(r)
Next Inner
$r.\mathbf{G} = s_c.\mathbf{G} \wedge ($
$r.T \leq s_c.T \wedge r.E < s_c.E$
$\vee d(r, s_c) < d(r, s_n))$
$\vee r.\mathbf{G} < s_c.\mathbf{G}$
else
else
$!Null(r) \vee$
$len(B_\mathbf{r}) > 0$
Join Tuples

Tasks:

1. **Oct**: The CC-Join as defined above is not lossless, i.e. since multiple join matches are aggregated there is a loss of information. Generalize the definition of the CC-Join for making optional the aggregation of multiple join matches
2. **Oct - Nov**: Starting from the provided *buffered* implementation, extend the implementation of the CC-Join operator in the DBMS of PostgreSQL for the case when no aggregation is required.
3. **Dec**: Implement the above algorithm, with the possibility to compute the join without any buffering effect, i.e. without any buffer and any aggregation set. *Backtracking* (MarkPosition and FetchRow) of the inner relation should be used when multiple outer tuples

share the same join matches.

4. **Dec**: Can you describe application use cases of the cc-join without aggregation in the context of the feed Data Warehouse? Domain experts may help you in this.
5. **Jan - Feb**: Evaluate the efficiency of your approach analytically and empirically. Depending on the outcome of activity 4, the Feed Data Warehouse or synthetic data Show your empirical results in graphs comparing the buffered and the unbuffered implementations of the operator with and without aggregation. What are the advantages / disadvantages of having a buffered implementation? What are the limits of such an approach? When the buffer gets large, volatile available memory may not be enough for storing it and tuples need to be rewritten on disc. Can you find a good trade-off between buffering and backtracing?
6. **Mar**: In your report define the problem and your solution precisely and design a representative running example to illustrate your approach. Describe your algorithm using pseudo code.
7. Present progress and plans once every week to your supervisor.

Optional tasks:

1. Implement the CC-Join using a hash-based approach

Literature:

[1] Y. N. Silva, W. G. Aref, and M. H. Ali. The similarity join database operator. ICDE, 2010.

Supervisor: Francesco Cafagna

Starting date: 4 October 2013

Ending date: 3th April 2014

Department of Informatics, University of Zurich

[illegible]

Prof. Dr. Michael Böhlen