Adaptive Join Algorithms for Distributed Databases

Prof. Vasilis Vassalos

Informatics Department
Athens University of Business and Economics
Content

1. Motivation

2. Problem Definition

3. Structure of Algorithms
   - Architecture
   - Stages
   - Key Points

4. Existing Algorithms

5. DINER
What is different compared to traditional databases?

- The input relations are provided by autonomous sources
- Data is transported through unreliable network environment
- Data arrival rate cannot be controlled
What is the impact on users?

- The complete data might be available after a long time.
- Traditional join algorithms are unusable, they produce partial results if the data is completely available.
- The availability of partial join results is important for a wide range of applications.
Performance is defined by:

- Producing results as soon as first input tuple is available
- Smoothing of join result production
- Masking source and network delay
- Performance criteria: higher join result rate
Problem definition

Given two relations streamed by remote sources and under limited memory constraints, the goal is to produce the correct join result at a high rate.
Content

1. Motivation

2. Problem Definition

3. Structure of Algorithms
   - Architecture
   - Stages
   - Key Points

4. Existing Algorithms

5. DINER
Algorithms’ Architecture

$S_A$

Input Buffer

$S_B$

Statistics

Index A

Index B

Tuple Storage

Disk$_A$

Disk$_B$
Online Phase
- Tuples arrive in the input buffer
- Tuples are processed in memory using an index
- Some statistics might be kept
- When memory is exhausted, some tuples are flushed on disk

Reactive Phase
- Is activated when both streams experience delay
- Part of the disk buffered data is processed

Completing Phase
- Is activated after the data has completely arrived
- Joins the rest of the disk buffered data
Content

1. Motivation

2. Problem Definition

3. Structure of Algorithms
   - Architecture
   - Stages
   - Key Points

4. Existing Algorithms

5. DINER
The key points of the algorithms

**The flushing policy** The tuples that don’t participate in joins should be flushed to disk

**Reactive phase** The reactive phase should be able to switch between offline and online processing as soon as new input arrives to avoid input buffer overflow
Double Pipeline Hash Join - DPHJ

- Among the first algorithms
- Uses a hash index for indexing in-memory tuples of each stream
- Inefficient flushing policy: the largest hash bucket pair from both relations is flushed
- Nonexistent reactive phase
**XJoin**

- Developed in the same time with DPHJ
- Uses a hash index for in-memory tuples which organizes the tuples in buckets
- Inefficient flushing policy: the largest bucket of one relation is flushed
- The reactive phase joins a disk bucket of one relation with the corresponding memory bucket of the opposite relation
Hash Merge Join - HMJ

- A hash based algorithm
- Introduces a more advanced flushing policy: the bucket pair that has the largest size and keeps memory space balanced between the two relations becomes victim.
- The reactive phase joins on disk data
Progressive Merge Join - RPJ

- A hash based algorithm
- The first algorithm to keep statistics on the number of results produced by each bucket
- Advanced flushing policy: flushes the bucket with the smallest probability to participate in joins
- Reactive phase aims to produce a big number of results
- Reactive phase is not responsive: it does not respond fast to new input
DINER vs Existing Algorithms

- Supports equijoins and range queries
- Introduces an intuitive flushing policy
  - Significantly higher result rate compared to existing algorithms
- Introduces a more responsive Reactive Phase
  - Takes advantage of source blocking and continues to produce results
  - Prompt hand over when new input tuples arrive
- A leaner algorithm - better performance in more constrained memory environment
Logical organization of in-memory tuples

```
  SA
  |   Up   |
  | Middle |
  | Low    |
  \-------------------
  LastUpValue_A

  SB
  |   Up   |
  | Middle |
  | Low    |
  \-------------------
  LastUpValue_B

  LastLwValue_A

  LastLwValue_B

LwJoins_B  The nr of join results
LwTuples_B  The number of tuples
BfLw_B      Benefit
Lw area
S_B stream

Bf{Up | Md | Lw}_{A|B} = \frac{\{Up | Md | Lw\}Joins_{A,B}}{\{Up | Md | Lw\}Tuples_{A,B}}

Online Phase
```
**Reactive Phase**

**Outer Relation**
- `joinedOuter`
- `outerSize` = total number of blocks of Outer
- In-memory batch of `outerMem` blocks
- Painted Areas Have Joined
- Currently joined blocks
- Swap Outer–Inner when entire Outer joins with Inner up to block `joinedInner`

**Inner Relation**
- `currInner`
- `joinedInner`
- Arrived after join with Outer started

**Outer Relation**
- Painted Areas Have Joined
- Currently joined blocks
- `joinedOuter`
- `outerSize` = total number of blocks of Outer
- In-memory batch of `outerBlocks` blocks

**Inner Relation**
- `currInner`
- `joinedInner`
- `outerSize` = total number of blocks of Outer