Parallel Query Processing
Contents

Concepts and goals of parallel query processing
Multiprocessor architectures
Parallel relational operators
Parallel query processing
Parallel database systems
  Brief list of existing machines and prototype systems
Concepts and goals

Main problem: I/O bottleneck

*transferring the data from disk to main memory*

Solution

*increase the I/O bandwidth through parallelism*
Ideal advantages of parallel systems

High performance

*short response time or total time*
*good load balancing among processors*

High availability

*handling failures of hardware elements*
*redundancy and consistency*

Extensibility

*more processing and storage power can be added*
Speedup and scaleup

Speedup

\[ \text{twice as much hardware results in half elapsed time} \]

Scaleup

\[ \text{twice as much hardware can perform twice as large a task in the same elapsed time} \]

Bottlenecks

Start-up

\[ \text{many processors } \Rightarrow \text{long start-up time} \]

Interference

\[ \text{more processors } \Rightarrow \text{more communications} \]

Data skew

\[ \text{it makes data distribution difficult} \]
Multiprocessor architectures

Shared-memory architectures
Shared-disk architectures
Shared-nothing architectures
Hybrid architectures
Shared-memory architectures

More processors - one memory

Characteristics

any CPU has access to any memory module or disk unit

Advantages  Disadvantages
simplicity    cost
load balancing little extensibility

low availability
Shared-disk architectures

More processors and memory units

Interconnection Network

- Proc. 1
  - Memory
- Proc. 2
  - Memory
- ...
- Proc. n
  - Memory

Characteristics

*any CPU has access to any disk unit but exclusive access to its main memory*

Advantages
- cost
- extensibility
- load balancing
- availability
- easy migration

Disadvantages
- higher complexity
- potential performance problems
Shared-nothing architectures

Exclusive access to memory and disk

Interconnection Network

Proc. 1
Memory

Proc. 2
Memory

... ...

Proc. n
Memory

Characteristics

any CPU has only exclusive access to its main memory and disk

Advantages

cost
extensibility
availability

Disadvantages

higher complexity
load balancing
Hybrid architectures

Goals:

combine the advantages of different architectures

use different processing elements in the system

The system is basically a shared-nothing architecture where each node can be a multicomputer of any architecture.
Parallel relational operators

Data partitioning

Parallelisation of relational operators

Join

Parallelisation of join

Pipelined hash-join
Data partitioning

Partitioning = distributing the tuples of a relation over several disks

Goal: allowing parallel databases to exploit the I/O bandwidth of multiple disks by reading and writing them in parallel

Relations are declustered (partitioned horizontally) based on a function:

- round-robin
- range index
- hash function
Round-robin partitioning

It maps the i’th tuple to disk i mod n

Processors and memories with interconnection network
Range index partitioning

*It clusters tuples with similar attributes together in the same partition*

Processors and memories with interconnection network

- disk 1: a - c
- disk 2: d - g
- ...: ...
- disk m: w - z
Hash partitioning

It maps each tuple to a location based on a hash function. Hashing tends to randomise data rather than cluster it.
Advantages of the partitioning strategies

**Round-robin**
- Sequential scan of all tuples in each query

**Range index**
- Sequential scan of all tuples in each query
- Associative search for data
- Clustering data

**Hash**
- Sequential scan of all tuples in each query
- Associative search for data
Problems with data partitioning

**data skew**

A query may work mainly on one partition because of the actual data placement \(\Rightarrow\) unbalanced load on the processors

**Dynamic reorganisation of relations**

With a given partitioning the criteria used for data placement can change to the extent that load balancing degrades significantly \(\Rightarrow\) the relation should be reorganised. This should be done on-line efficiently transparently to the users and compiled queries
Parallel relational operators

Relational algebra allows parallel processing due to its properties

set-oriented processing

simple operations

limited number of operations

simple improvements have substantial effect (large volume of data)

relatively independent of hardware architecture
Parallel relational operators

Basic idea

use parallel data streams instead of writing new parallel operators ⇒ use existing sequential relational operators in parallel

Solution

each relational operator has a set of input ports on which input tuples arrive and an output port to which the operator’s output stream is sent

The parallel dataflow works by partitioning and merging data streams into these sequential ports
From sequential to parallel execution

```sql
insert into C
select *
from A, B
where A.x = B.y;
```
Extending relational algebra for parallel query processing

Relational operations
   (select, project, cp, join, union, diff, inters)

Splitting of results over multiple output streams

Operands consisting of multiple input streams

Explicit allocation of processes to processors
Parallelisation of join

Splitting is consistent if

\[ R_i \bowtie S_j = \emptyset \quad \text{if} \quad i \neq j \]
Pipelined hash-join

Goal: produce output tuples as early as possible
Parallel query processing

= automatic translation of a query into an efficient execution plan
  + its parallel execution

The translation must be correct

  *the execution of the plan produces the expected result*

The execution plan must be optimal

  *in that it minimises a cost function that captures resource consumption*
Parallel query processing

**JOQR: Join Ordering and Query Rewrite**
Parallel query processing

1. translation
   of the relational algebra expression to a query tree

2. optimisation
   reordering of join operations in the query tree and choose among different join algorithms to minimise the cost of the execution

3. parallelisation
   transforming the query tree to a physical operator tree and loading the plan to the processors

4. execution
   running the concurrent transactions
Translation to query tree

Relational query $\Rightarrow$ (rewrite) $\Rightarrow$ query tree

SELECT P.name
FROM P, R, S
WHERE S.thema LIKE 'Computing'
AND S.class = R.class
AND R.id = P.id

Query Tree

$\text{proj P.name}$

$\text{sel S.thema}$

$\text{Query Tree}$

$\text{P}$

$\text{R}$

$\text{S}$
Optimisation

Reordering of the join operations and choosing an evaluation algorithm for each join operation

\[
\begin{align*}
\text{proj } & \quad \text{P.name} \\
\text{sel } & \quad \text{S.thema} \\
\text{P} & \\
\text{R} & \\
\text{S} & \\
\text{P} & \\
\text{simple-hash} & \\
\text{P.id = right.id} & \\
\text{R.class = S.class} & \\
\text{sel } & \quad \text{S.thema} \\
\text{S} & \\
\text{index-scan} & \\
\text{R} & \\
\text{P} & \\
\text{clustered index-scan} & \\
\end{align*}
\]
Parallelisation of the query

1. Extract parallelism by macro-expansion of the annotated query tree to an operator tree

   The operator tree identifies

   atomic code pieces (operators)

   timing constraints between the operators

2. Schedule the operator tree on the parallel machine

   Goal: allocate machine resources so as to exploit the available parallelism while respecting timing and data placement constraints.
Parallelisation of the query

Physical operator tree

s1 = sel S.theme
pj1 = proj S.class
pj2 = proj P.id, P.name
pj3 = proj R.id
pj4 = proj P.name
b1, b2: build hash-table
p1, p2: probe hash-table
Constraints on available parallelism

Precedence constraint

*timing constraints between operators*
*e.g. hash table must be built completely before probing*

Parallel constraint

*piping the output of an operator to the input of another one* ⇒
*eliminating the need of buffering intermediate results*

Data placement constraint

*in shared-nothing systems the scan operations must be localised to specific processors that can access the relation*
Parallel execution control strategies

Control flow

a single control node controls all processes related to a query
it starts all processes and synchronise them
scheduling is performed by the control node

Data flow

no centralised control
processes on different nodes trigger each other with data messages
data driven execution: if enough input is available the process starts automatically
Parallel execution control strategies

Control flow is the traditional approach

Advantages of data flow control
- reduces the control messages transferred through the network
- reduces the workload of a particular node (the control node)
- provides pipelining naturally

Disadvantages of data flow control
- it means more asynchronous activity
- more competition for a processor
- higher protocol complexity
- providing data allocation information to all nodes is difficult
# Parallel Database Systems

## Research systems and projects

<table>
<thead>
<tr>
<th>Gamma</th>
<th>Bubba</th>
<th>Prisma/DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>HC16-186</td>
<td>XPRS project</td>
<td>Volcano project</td>
</tr>
</tbody>
</table>

## Commercial systems

<table>
<thead>
<tr>
<th>Oracle Parallel Server</th>
<th>Sybase MPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Informix Online XPS</td>
<td>IBM DB2 Parallel Edition</td>
</tr>
<tr>
<td>NCR Teradata DBS</td>
<td>Tandem NonStop SQL</td>
</tr>
</tbody>
</table>
Gamma

University of Wisconsin (David DeWitt)

hypercube, 32 nodes

horizontal partition

hash-based parallel algorithms for operations

data flow query execution (no pipelining)

hybrid hash-join

ported to an Intel iPSC/2 hypercube with 32 nodes

Bubba

MCC, Austin

never completely implemented

OLTP and DSS load

horizontal data partition over all disks

compiled PFAD, a parallel, set-oriented execution model

data flow query execution (no pipelining)

operations take place where data is
PRISMA/DB

University of Twente  
(Peter M.G. Apers)

main-memory parallel  
database system

shared-nothing architecture

data flow query execution  
(with pipelining)

simple, grace and hybrid  
hash-join

HC16-186

Trondheim, Norway

hypercube intrerconnect,  
16 nodes

not a fullfledge DBMS

horizontal partition of the data  
over all disks

redistribution of data
XPRS

M. Stonebraker

Shared-memory system

Based on the Postgres next-generation DBMS

intra- and inter-operator parallelism

high performance and availability

Volcano project

Götz Graefe

Extensible and parallel query evaluation system

supports shared-memory, shared-nothing and hybrid architectures

provides a rich environment for research and education

data flow query execution
Commercial parallel database systems (brief)

Architecture

SMP, Symmetric Multiprocessing (shared memory)
Clustered SMP
MPP, Massively Parallel Processors (shared nothing)

Oracle Parallel Server
shared-disk
parallel cache management

Sybase MPP
shared-nothing (MPP)
partitioned database
Commercial parallel database systems (brief)

- **Informix Online XPS**
  - combination of shared-memory and shared-nothing architecture
  - partitioned database

- **Tandem NonStop SQL**
  - primarily designed for OLTP transactions
  - shared-nothing architecture

- **NCR Teradata DBS**
  - first commercial system
  - shared-nothing architecture
  - installed on systems with more than 100 processors

- **IBM DB2 Parallel Edition**
  - shared-nothing architecture