Parallel Database Systems:
The Future of High Performance Database Processing

David J. DeWin\textsuperscript{2}
Computer Sciences Department
University of Wisconsin
1210 W. Dayton St.
Madison, WI 53706
dewitt@cs.wisc.edu

Jim Gray
San Francisco Systems Center
Digital Equipment Corporation
455 Market St. 7th floor
San Francisco, CA. 94105-2403
Gray@SFbayenet.dec.com

January 1992

Abstract: Parallel database machine architectures have evolved from the use of exotic hardware to a software parallel dataflow architecture based on conventional shared-nothing hardware. These new designs provide impressive speedup and scaleup when processing relational database queries. This paper reviews the techniques used by such systems, and surveys current commercial and research systems.

1. Introduction

Highly parallel database systems are beginning to displace traditional mainframe computers for the largest database and transaction processing tasks. The success of these systems refutes a 1983 paper predicting the demise of database machines [BORAI]. Ten years ago the future of highly-parallel database machines seemed gloomy, even to their staunchest advocates. Most database machine research had focused on specialized, often trendy, hardware such as CCD memories, bubble memories, head-per-track disks, and optical disks. None of these technologies fulfilled their promises; so there was a sense that conventional cpus, electronic RAM, and moving-head magnetic disks would dominate the scene for many years to come. At that time, disk throughput was predicted to double while processor speeds were predicted to increase by much larger factors. Consequently, critics predicted that multiprocessor systems would soon be I/O limited unless a solution to the I/O bottleneck was found.

While these predictions were fairly accurate about the future of hardware, the critics were certainly wrong about the overall future of parallel database systems. Over the last decade Teradata, Tandem, and a host of startup companies have successfully developed and marketed highly parallel database machines.
Agenda

Parallelism metrics
Parallel architecture
Parallel OLAP operators
Cloud parallel database
Agenda

Parallelism metrics
Parallel architecture
Parallel OLAP operators
Cloud parallel database
Parallel Database History

1980’s: database machines
  • Specialized hardware to make databases run fast
  • Special hardware cannot catch up with Moore’s Law

1980’s – 2010’s: shared-nothing architecture
  • Connecting machines using a network

2010’s – future?
Scaling in Parallel Systems

**Linear speedup**

- Twice as much hardware can perform the task in half the elapsed time
- Speedup = \( \frac{\text{small system elapsed time}}{\text{big system elapsed time}} \)
- Linear speedup = N, where the big system is N times larger than the small system
Scaling in Parallel Systems

**Linear speedup**
- Twice as much hardware can perform the task in half the elapsed time
- Speedup = \( \frac{\text{small system elapsed time}}{\text{big system elapsed time}} \)
- Linear speedup = \( N \), where the big system is \( N \) times larger than the small system

**Linear scaleup**
- Twice as much hardware can perform twice as large a task in the same elapsed time
- Scaleup = \( \frac{\text{small system elapsed time on small problem}}{\text{big system elapsed time on big problem}} \)
- Linear scaleup = 1
Scaling in Parallel Systems

The Good Speedup Curve

Ideal speedup
Scaling in Parallel Systems

**Ideal speedup**

**No speedup**
Scaling in Parallel Systems

Ideal speedup

No speedup

In practice

A Bad Speedup Curve
No Parallelism

A Bad Speedup Curve
3-Factors

Start-up
Interference
Skew
Threats to Parallelism

- Ideal
- non-ideal

Startup

processors & disks

Start parallel tasks

Collect results

Starting remote tasks incurs performance overhead
Threats to Parallelism

Examples of interference

- Shared hardware resources (e.g., memory, disk, network)
- Synchronization (e.g., locking)
Threats to Parallelism

Some nodes take more time to execute the assigned tasks, e.g.,
- More tasks assigned
- More computational intensive tasks assigned
- Node has slower hardware
Agenda

Parallelism metrics

Parallel architecture

Parallel OLAP operators

Cloud parallel database
Design Spectrum

Shared Memory

Shared Disk

Shared Nothing
Design Spectrum – Shared Memory (SM)

All processors share direct access to a common global memory and to all disks
  • Does not scale beyond a single server

Example: multicore processors
Design Spectrum – Shared Disk (SD)

Each processor has a private memory but has direct access to all disks
  • Does not scale beyond tens of servers

Example: Network attached storage (NAS) and storage area network (SAN)
Each memory and disk is owned by some processor that acts as a server for that data

- Scales to **thousands of servers and beyond**

Important optimization goal: minimize network data transfer
Agenda

Parallelism metrics
Parallel architecture
Parallel OLAP operators
Cloud parallel database
Old uni-processor software must be rewritten to benefit from parallelism

Most database programs are written in relational language SQL
  • **Can make SQL work on parallel hardware without rewriting**
  • Benefits of a high-level programming interface
How to Build Parallel Database?

Old uni-processor software must be rewritten to benefit from parallelism.

Most database programs are written in relational language SQL

- Can make SQL work on parallel hardware without rewriting
- Benefits of a high-level programming interface

Pipelined Parallelism

Partitioned Parallelism
Pipelined Parallelism

Pipelined parallelism: pipeline of operators

Source Data → Scan → Sort → Processor 1 → Processor 2
Pipelined Parallelism

Pipelined parallelism: pipeline of operators

**Advantages**
- Avoid writing intermediate results back to disk
Pipelined Parallelism

Pipelined parallelism: pipeline of operators

**Advantages**
- Avoid writing intermediate results back to disk

**Disadvantages**
- Small number of stages in a query
- Blocking operators: e.g., sort and aggregation
- Different speed: scan faster than join. Slowest operator becomes the bottleneck
Partitioned Parallelism

Map tuple $i$ to disk ($i \mod n$)

- **Advantage**: Simplicity, good load balancing
- **Disadvantage**: Hard to identify the partition of a particular record

Round robin
Partitioned Parallelism

Map contiguous attribute ranges to partitions

- **Advantage**: Good locality due to clustering
- **Disadvantage**: May suffer from skewness
Map based on the hash value of tuple attributes

- **Advantage**: Good load balance, low skewness
- **Disadvantage**: Bad locality
Parallel data streams so that sequential operator code is not modified

- Each operator has a set of input and output ports
- Partition and merge these ports to sequential ports so that **an operator is not aware of parallelism**
Parallel data streams so that sequential operator code is not modified

- Each operator has a set of input and output ports
- Partition and merge these ports to sequential ports so that an operator is not aware of parallelism
Parallelism within Relational Operators

Parallel data streams so that sequential operator code is not modified

- Each operator has a set of input and output ports
- Partition and merge these ports to sequential ports so that an operator is not aware of parallelism

```
insert into C
select *
from A, B
where A.x = B.y;
```
Data Shuffle

Single-node query plan

Distributed query plan

\[ R \bowtie S \]

\[ R \bowtie S \]

Exchange  Exchange
Data Shuffle – Example

Site 1
- R₁
- S₁

Site 2
- S₂

Site 3
- R₃
- S₃

Query plan:
- Exchange
- Exchange
- R
- S
Data Shuffle – Single-Site

Solution 1: send all the involved tables to a single site

- **Advantage**: Single-site query execution is a solved problem
- **Disadvantage**: (1) Single site execution can be slow (2) Data may not fit in single site’s memory or disk
Solution 2: Keep one relation partitioned and broadcast the other relation to all sites

- **Advantage**: One relation does not need to move
- **Disadvantage**: Still need to broadcast the other relation to all sites
Data Shuffle – Co-partition

Solution 3: Partition both relations using the join key

- **Advantage**: Each site has less data to process
- **Disadvantage**: Both relations are shuffled (if not already partitioned based on join key)
Specialized Parallel Operators

Semi-join

- Example:

```sql
SELECT *
FROM T1, T2
WHERE T1.A = T2.C
```

Agenda

Parallelism metrics
Parallel architecture
Parallel OLAP operators
Cloud parallel database
Paradigm Shift in Architecture — Disaggregation

Storage Disaggregation vs. Shared Nothing vs. Shared Disk
Paradigm Shift in Architecture — Disaggregation

Storage Disaggregation

VS.

Shared Nothing

Shared Disk

Feature 1: All compute nodes can access the entire storage service
Feature 2: Can perform limited computation in the storage service
Feature 3: The storage service is highly available
Paradigm Shift in Architecture — Disaggregation

Storage Disaggregation

Key challenge: **Network becomes a bottleneck**
- Performance of disaggregation can be 10x lower than shared-nothing [1]

More on this topic in a few lectures

Q/A – Parallel Database

How is data skew handled by hash partitioning?

Is partitioning part of the SQL interface or is it hidden from users?

The paper mentioned that SM and SD systems failed to scale well because of limited network bandwidth; Is this still true today?

No quantitative comparison of scalability

How to cope with locking in a shared-nothing database?
Before Next Lecture

Submit review for