CSE 544: Principles of Database Systems

Anatomy of a DBMS, Parallel Databases
Announcements

• Lecture on Thursday, May 2nd:
  – Moved to 9am-10:30am, CSE 403

• Paper reviews:
  – Anatomy paper was due yesterday; will discuss today in class
  – Join processing paper due Wednesday

• HW2 is due on Monday, May 6th
  – Lots of work! You should have already started!
Overview of Today’s Lecture

• Discuss in class the Anatomy paper

• Parallel databases (Chapter 22.1 – 22.5)
DMBS Architecture: Outline

• Main components of a modern DBMS
• Process models
• Storage models
• Query processor
DBMS Architecture

Process Manager
- Admission Control
- Connection Mgr

Query Processor
- Parser
- Query Rewrite
- Optimizer
- Executor

Storage Manager
- Access Methods
- Buffer Manager
- Log Manager
- Lock Manager

Shared Utilities
- Memory Mgr
- Disk Space Mgr
- Replication Services
- Admin Utilities

DMBS Architecture: Outline

• Main components of a modern DBMS
• Process models
• Storage models
• Query processor
Q: Why not simply queue all user requests, and serve them one at the time?
Q: Why not simply queue all user requests, and serve them one at the time?
A: Because of the high disk I/O latency
   Corollary: in a main memory db you can service transactions sequentially!

Alternatives
1. **Process per connection**
2. **Server process** (thread per connection)
   - OS threads or DBMS threads
3. **Server process with I/O process**
Process Per Connection

• **Overview**
  – DB server forks one process for each client connection

• **Advantages**
  – ?

• **Drawbacks**
  – ?
Process Per Connection

• **Overview**
  – DB server forks one process for each client connection

• **Advantages**
  – Easy to implement (OS time-sharing, OS isolation, debuggers, etc.)

• **Drawbacks**
  – Need OS-supported “shared memory” (for lock table, buffer pool)
  – Not scalable: memory overhead and expensive context switches
Server Process

• Overview
  – Dispatcher thread listens to requests, dispatches worker threads

• Advantages
  – ?
  – ?

• Drawbacks
  – ?
Server Process

• **Overview**
  – *Dispatcher* thread listens to requests, dispatches *worker* threads

• **Advantages**
  – Shared structures can simply reside on the heap
  – Threads are lighter weight than processes: memory, context switching

• **Drawbacks**
  – Concurrent programming is hard to get right (race conditions, deadlocks)
  – Subtle API thread differences across different operating systems make portability difficult
Sever Process with I/O Process

Problem: entire process blocks on synchronous I/O calls

- **Solution 1:** Use separate process(es) for I/O tasks
- **Solution 2:** Modern OS provide asynchronous I/O
DBMS Threads vs OS Threads

• Why do DBMSs implement their own threads?
DBMS Threads vs OS Threads

• **Why do DBMSs implement their own threads?**
  – Legacy: originally, there were no OS threads
  – Portability: OS thread packages are not completely portable
  – Performance: fast task switching

• **Drawbacks**
  – Replicating a good deal of OS logic
  – Need to manage thread state, scheduling, and task switching

• **How to map DBMS threads onto OS threads or processes?**
  – Rule of thumb: one OS-provided dispatchable unit per physical device
  – See page 9 and 10 of Hellerstein and Stonebraker’s paper
Historical Perspective (1981)

In 1981:

- No OS threads
- No shared memory between processes
  - Makes one process per user hard to program
- Some OSs did not support many to one communication
  - Thus forcing the one process per user model
- No asynchronous I/O
  - But inter-process communication expensive
  - Makes the use of I/O processes expensive

- Common original design: DBMS threads, frequently yielding control to a scheduling routine
Commercial Systems

• **Oracle**
  – Unix default: process-per-user mode
  – Unix: DBMS threads multiplexed across OS processes
  – Windows: DBMS threads multiplexed across OS threads

• **DB2**
  – Unix: process-per-user mode
  – Windows: OS thread-per-user

• **SQL Server**
  – Windows default: OS thread-per-user
  – Windows: DBMS threads multiplexed across OS threads
DMBS Architecture: Outline

• Main components of a modern DBMS
• Process models
• Storage models
• Query processor
Storage Model

- **Problem**: DBMS needs spatial and temporal control over storage
  - Spatial control for performance
  - Temporal control for correctness and performance

- **Alternatives**
  - Use “raw” disk device interface directly
  - Use OS files
Spatial Control
Using “Raw” Disk Device Interface

• **Overview**
  – DBMS issues low-level storage requests directly to disk device

• **Advantages**
  – ?
  – ?

• **Disadvantages**
  – ?
Spatial Control
Using “Raw” Disk Device Interface

• **Overview**
  – DBMS issues low-level storage requests directly to disk device

• **Advantages**
  – DBMS can ensure that important queries access data sequentially
  – Can provide highest performance

• **Disadvantages**
  – Requires devoting entire disks to the DBMS
  – Reduces portability as low-level disk interfaces are OS specific
  – Many devices are in fact “virtual disk devices”
    • SAN = storage area network; NAS = network attached device
Spatial Control Using OS Files

• **Overview**
  – DBMS creates one or more very large OS files

• **Advantages**
  – ?

• **Disadvantages**
  – ?
Spatial Control Using OS Files

• **Overview**
  – DBMS creates one or more very large OS files

• **Advantages**
  – Allocating large file on empty disk can yield good physical locality

• **Disadvantages**
  – Must control the timing of writes for *correctness* and *performance*
  – OS may further delay writes
  – OS may lead to double buffering, leading to unnecessary copying
  – DB must fine tune when the log tail is flushed to disk
Historical Perspective (1981)

• Recognizes mismatch problem between OS files and DBMS needs
  – If DBMS uses OS files and OS files grow with time, blocks get scattered
  – OS uses tree structure for files but DBMS needs its own tree structure

• Other proposals at the time
  – Extent-based file systems
  – Record management inside OS
Commercial Systems

- Most commercial systems offer both alternatives
  - Raw device interface for peak performance
  - OS files more commonly used

- In both cases, we end-up with a DBMS file abstraction implemented on top of OS files or raw device interface
Temporal Control
Buffer Manager

• Correctness problems
  – DBMS needs to control when data is written to disk in order to provide **transactional semantics** (we will study transactions later)
  – OS buffering can **delay writes**, causing problems when crashes occur

• Performance problems
  – OS optimizes buffer management for general workloads
  – DBMS understands its workload and can do better
  – Areas of possible optimizations
    • Page replacement policies
    • Read-ahead algorithms (physical vs logical)
    • Deciding when to flush tail of write-ahead log to disk
Historical Perspective (1981)

- Problems with OS buffer pool management long recognized
  - Accessing OS buffer pool involves an expensive system call
  - Faster to access a DBMS buffer pool in user space
  - LRU replacement does not match DBMS workload
  - DBMS can do better
    - OS can do only sequential prefetching, DBMS knows which page it needs next and that page may not be sequential
    - DBMS needs ability to control when data is written to disk
Commercial Systems

• DBMSs implement their own buffer pool managers

• Modern filesystems provide good support for DBMSs
  – Using large files provides good spatial control
  – Using interfaces like the mmap suite
    • Provides good temporal control
    • Helps avoid double-buffering at DBMS and OS levels
DMBS Architecture: Outline

- Main components of a modern DBMS
- Process models
- Storage models
- Query processor
Query Processor

1. Parsing and Authorization
   - Catalog management
2. Query rewrite
   - View inlining, etc
3. Optimizer
   - System R v.s. Volcano/Cascades style
   - Selectivity estimation
4. Query execution
   - Iterator model: init(), get_next(), close()
   - What is the “Halloween problem”? 
5. Access methods
   - Pass a search predicate (SARG) to init()
The “prepare” statement must choose a plan without knowing the actual predicate values. Discuss the *Anatomy* paper.

Figure 1: Plan diagram for TPC-H Query 8
Parallel Databases
Parallel v.s. Distributed Databases

• Parallel database system:
  – Improve performance through parallel implementation
  – Will discuss in class

• Distributed database system:
  – Data is stored across several sites, each site managed by a DBMS capable of running independently
  – Will not discuss in class
Parallel DBMSs

• **Goal**
  – Improve performance by executing multiple operations in parallel

• **Key benefit**
  – Cheaper to scale than relying on a single increasingly more powerful processor

• **Key challenge**
  – Ensure overhead and contention do not kill performance
Performance Metrics for Parallel DBMSs

• **Speedup**
  - More processors ➔ higher speed

• **Scaleup**
  - More processors ➔ can process more data

• **Batch scaleup/speedup**
  - Decision Support: individual query should run faster (speedup) or same speed (scaleup)

• **Transaction scaleup/speedup**
  - OLTP: Transactions Per Second (TPS) should increase (speedup) or should stay constant (scaleup)
Linear v.s. Non-linear Speedup

Speedup

# processors (=P)
Linear v.s. Non-linear Scaleup

Batch Scaleup

\[
\begin{array}{c}
\times 1 \\
\times 5 \\
\times 10 \\
\times 15 \\
\end{array}
\]

# processors (=P) AND data size
Challenges to Linear Speedup and Scaleup

• **Startup cost**
  – Cost of starting an operation on many processors

• **Interference**
  – Contention for resources between processors

• **Skew**
  – Slowest processor becomes the bottleneck
Architectures for Parallel Databases

- Shared memory
- Shared disk
- Shared nothing
Shared Memory

Interconnection Network

Global Shared Memory

P  P  P

D  D  D
Shared Disk

Interconnection Network
Shared Nothing

Interconnection Network

P
M
D

P
M
D

P
M
D
Shared Nothing

• Most scalable architecture
  – Minimizes interference by minimizing resource sharing
  – Can use commodity hardware
  – Terminology: processor = server = node
  – P = number of nodes

• Also most difficult to program and manage
Taxonomy

- **Inter-query parallelism**
  - Transaction per node
  - OLTP

- **Inter-operator parallelism**
  - Operator per node
  - Both OLTP and Decision Support

- **Intra-operator parallelism**
  - Operator on multiple nodes
  - Decision Support

We study only intra-operator parallelism: most scalable
Review in Class

Basic query processing on one node.

Given relations R(A,B) and S(B, C), compute:

- Selection: $\sigma_{A=123}(R)$
- Group-by: $\gamma_{A,\text{sum}(B)}(R)$
- Join: $R \bowtie S$
Horizontal Data Partitioning

• Partition a table \( R(K, A, B, C) \) into \( P \) chunks \( R_1, \ldots, R_P \), stored at the \( P \) nodes

• **Block Partition**: \( \text{size}(R_1) \approx \ldots \approx \text{size}(R_P) \)

• **Hash partitioned on attribute \( A \)**: 
  – Tuple \( t \) goes to chunk \( i = (h(t.A) \mod P) + 1 \)

• **Range partitioned on attribute \( A \)**: 
  – Partition the range of \( A \) into \( -\infty = v_0 < v_1 < \ldots < v_P = \infty \) 
  – Tuple \( t \) goes to chunk \( i \), if \( v_{i-1} \leq t.A < v_i \)
Parallel GroupBy

\( R(K,A,B,C) \), discuss in class how to compute these GroupBy’s, for each of the partitions

- \( Y_{A,sum(C)}(R) \)

- \( Y_{B,sum(C)}(R) \)
Parallel GroupBy

\[ Y_{A, \text{sum}(C)}(R) \]

- If \( R \) is partitioned on \( A \), then each node computes the group-by locally.
- Otherwise, hash-partition \( R(K, A, B, C) \) on \( A \), then compute group-by locally:

Reshuffle \( R \) on attribute.
Speedup and Scaleup

• The runtime is dominated by the time to read the chunks from disk, i.e. size($R_i$)

• If we double the number of nodes $P$, what is the new running time of $\gamma_{A,\text{sum}(C)}(R)$?

• If we double both $P$ and the size of the relation $R$, what is the new running time?
Uniform Data v.s. Skewed Data

• **Uniform partition:**
  - $\text{size}(R_1) \approx \ldots \approx \text{size}(R_P) \approx \frac{\text{size}(R)}{P}$
  - Linear speedup, constant scaleup

• **Skewed partition:**
  - For some $i$, $\text{size}(R_i) \gg \frac{\text{size}(R)}{P}$
  - Speedup and scaleup will suffer
Uniform Data v.s. Skewed Data

Let $R(K,A,B,C)$; which of the following partition methods may result in skewed partitions?

- **Block partition**
- **Hash-partition**
  - On the key $K$
  - On the attribute $A$
- **Range-partition**
  - On the key $K$
  - On the attribute $A$
Uniform Data v.s. Skewed Data

Let \( R(K,A,B,C) \); which of the following partition methods may result in skewed partitions?

- **Block partition**
- **Hash-partition**
  - On the key \( K \)
  - On the attribute \( A \)
- **Range-partition**
  - On the key \( K \)
  - On the attribute \( A \)

Uniform	Uniform

May be skewed

E.g. when all records have the same value of the attribute \( A \), then all records end up in the same partition.

Difficult to partition the range of \( A \) uniformly.
Parallel Join

• In class: compute $R(A, B) \bowtie S(B, C)$

$$R_1, S_1 \quad R_2, S_2 \quad \ldots \quad R_P, S_P$$
Parallel Join

• In class: compute $R(A,B) \bowtie S(B,C)$

Reshuffle $R$ on $R.B$ and $S$ on $S.B$

Each server computes the join locally
Parallel Query Plans

- Same relational operators

- Add special split and merge operators
  - Handle data routing, buffering, and flow control

- Example: exchange operator
  - Inserted between consecutive operators in the query plan