Chapter 20: Database System Architectures
Chapter 21: Database System Architectures

▪ Centralized Database Systems
▪ Server System Architectures
▪ Parallel Systems
▪ Distributed Systems
▪ Network Types
Centralized Database Systems

- Run on a single computer system

**Single-user system**
  - Embedded databases

**Multi-user systems** also known as server systems.
  - Service requests received from client systems
  - Multi-core systems with *coarse-grained parallelism*
    - Typically a few to tens of processor cores
    - In contrast, fine-grained parallelism uses very large number of computers
Server System Architecture

- Server systems can be broadly categorized into two kinds:
  - **transaction servers**
    - Widely used in relational database systems, and
  - **data servers**
    - Parallel data servers used to implement high-performance transaction processing systems
Transaction Servers

- Also called **query server** systems or **SQL server** systems
  - Clients send requests to the server
  - Transactions are executed at the server
  - Results are shipped back to the client.
- Requests are specified in SQL, and communicated to the server through a *remote procedure call* (RPC) mechanism.
- Transactional RPC allows many RPC calls to form a transaction.
- Applications typically use ODBC/JDBC APIs to communicate with transaction servers
Transaction System Processes (Cont.)

- user process
  - ODBC
    - server process
  - JDBC
    - server process
- user process
- user process

buffer pool
- query plan cache
- log buffer
- lock table

- process monitor process
- lock manager process
- log writer process
- checkpoint process
- database writer process

- log disks
- data disks

shared memory
Transaction Server Process Structure

- A typical transaction server consists of multiple processes accessing data in shared memory

- Shared memory contains shared data
  - Buffer pool
  - Lock table
  - Log buffer
  - Cached query plans (reused if same query submitted again)

- All database processes can access shared memory

- Server processes
  - These receive user queries (transactions), execute them and send results back
  - Processes may be multithreaded, allowing a single process to execute several user queries concurrently
  - Typically multiple multithreaded server processes
Transaction Server Processes (Cont.)

- Database writer process
  - Output modified buffer blocks to disks continually
- Log writer process
  - Server processes simply add log records to log record buffer
  - Log writer process outputs log records to stable storage.
- Checkpoint process
  - Performs periodic checkpoints
- Process monitor process
  - Monitors other processes, and takes recovery actions if any of the other processes fail
    - E.g. aborting any transactions being executed by a server process and restarting it
Transaction System Processes (Cont.)

- Lock manager process
  - To avoid overhead of interprocess communication for lock request/grant, each database process operates directly on the lock table
    - instead of sending requests to lock manager process
  - Lock manager process still used for deadlock detection

- To ensure that no two processes are accessing the same data structure at the same time, databases systems implement *mutual exclusion* using either
  - Atomic instructions
    - Test-And-Set
    - Compare-And-Swap (CAS)
  - Operating system semaphores
    - Higher overhead than atomic instructions
Atomic Instructions

- **Test-And-Set**(M)
  - Memory location M, initially 0
  - Test-and-set(M) sets M to 1, and returns old value of M
    - Return value 0 indicates process has acquired the mutex
    - Return value 1 indicates someone is already holding the mutex
      - Must try again later
    - Release of mutex done by setting M = 0

- **Compare-and-swap**(M, V1, V2)
  - Atomically do following
    - If M = V1, set M = V2 and return success
    - Else return failure
  - With M = 0 initially, CAS(M, 0, 1) equivalent to test-and-set(M)
  - Can use CAS(M, 0, id) where id = thread-id or process-id to record who has the mutex
Data Servers/Data Storage Systems

- Data items are shipped to clients where processing is performed
- Updated data items written back to server
- Earlier generation of data servers would operate in units of data items, or pages containing multiple data items
- Current generation data servers (also called data storage systems) only work in units of data items
  - Commonly used data item formats include JSON, XML, or just uninterpreted binary strings
Data Servers/Storage Systems (Cont.)

- **Prefetching**
  - Prefetch items that may be used soon

- **Data caching**
  - Cache coherence

- **Lock caching**
  - Locks can be cached by client across transactions
  - Locks can be **called back** by the server

- **Adaptive lock granularity**
  - Lock granularity escalation
    - switch from finer granularity (e.g. tuple) lock to coarser
  - Lock granularity de-escalation
    - Start with coarse granularity to reduce overheads, switch to finer granularity in case of more concurrency conflict at server
  - Details in book
Data Servers (Cont.)

- **Data Caching**
  - Data can be cached at client even in between transactions
  - But check that data is up-to-date before it is used (**cache coherency**)
  - Check can be done when requesting lock on data item

- **Lock Caching**
  - Locks can be retained by client system even in between transactions
  - Transactions can acquire cached locks locally, without contacting server
  - Server **calls back** locks from clients when it receives conflicting lock request. Client returns lock once no local transaction is using it.
  - Similar to lock callback on prefetch, but across transactions.
Parallel Systems

- Parallel database systems consist of multiple processors and multiple disks connected by a fast interconnection network.
- Motivation: handle workloads beyond what a single computer system can handle
- High performance transaction processing
  - E.g. handling user requests at web-scale
- Decision support on very large amounts of data
  - E.g. data gathered by large web sites/apps
Parallel Systems (Cont.)

- A **coarse-grain parallel** machine consists of a small number of powerful processors.
- A **massively parallel** or **fine grain parallel** machine utilizes thousands of smaller processors.
  - Typically hosted in a **data center**
- Two main performance measures:
  - **throughput** --- the number of tasks that can be completed in a given time interval
  - **response time** --- the amount of time it takes to complete a single task from the time it is submitted
Speed-Up and Scale-Up

- **Speedup**: a fixed-sized problem executing on a small system is given to a system which is $N$-times larger.
  - Measured by:
    - $speedup = \frac{small\ system\ elapsed\ time}{large\ system\ elapsed\ time}$
    - Speedup is linear if equation equals $N$.

- **Scaleup**: increase the size of both the problem and the system
  - $N$-times larger system used to perform $N$-times larger job
  - Measured by:
    - $scaleup = \frac{small\ system\ small\ problem\ elapsed\ time}{big\ system\ big\ problem\ elapsed\ time}$
    - Scale up is linear if equation equals 1.
Scaleup

\[ \frac{T_S}{T_L} \]

linear scaleup

sublinear scaleup

problem size
Batch and Transaction Scaleup

- **Batch scaleup:**
  - A single large job; typical of most decision support queries and scientific simulation.
  - Use an $N$-times larger computer on $N$-times larger problem.

- **Transaction scaleup:**
  - Numerous small queries submitted by independent users to a shared database; typical transaction processing and timesharing systems.
  - $N$-times as many users submitting requests (hence, $N$-times as many requests) to an $N$-times larger database, on an $N$-times larger computer.
  - Well-suited to parallel execution.
Factors Limiting Speedup and Scaleup

Speedup and scaleup are often sublinear due to:

- **Startup/sequential costs**: Cost of starting up multiple processes, and sequential computation before/after parallel computation
  - May dominate computation time, if the degree of parallelism is high
  - Suppose $p$ fraction of computation is sequential
  - **Amdahl’s law**: speedup limited to: $1 / [(1-p)+(p/n)]$
  - **Gustafson’s law**: scaleup limited to: $1 / [n(1-p)+p]$

- **Interference**: Processes accessing shared resources (e.g., system bus, disks, or locks) compete with each other, thus spending time waiting on other processes, rather than performing useful work.

- **Skew**: Increasing the degree of parallelism increases the variance in service times of parallelly executing tasks. Overall execution time determined by **slowest** of parallelly executing tasks.
Interconnection Network Architectures

- **Bus.** System components send data on and receive data from a single communication bus;
  - Does not scale well with increasing parallelism.

- **Mesh.** Components are arranged as nodes in a grid, and each component is connected to all adjacent components
  - Communication links grow with growing number of components, and so scales better.
  - But may require $2\sqrt{n}$ hops to send message to a node (or $\sqrt{n}$ with wraparound connections at edge of grid).

- **Hypercube.** Components are numbered in binary; components are connected to one another if their binary representations differ in exactly one bit.
  - $n$ components are connected to $\log(n)$ other components and can reach each other via at most $\log(n)$ links; reduces communication delays.

- **Tree-like Topology.** Widely used in data centers today
Interconnection Architectures

(a) bus

(b) ring

(c) mesh

(d) hypercube
Interconnection Network Architectures

- **Tree-like or Fat-Tree Topology**: widely used in data centers today
  - Top of rack switch for approx 40 machines in rack
  - Each top of rack switch connected to multiple aggregation switches.
  - Aggregation switches connect to multiple core switches.
- **Data center fabric**

(e) tree-like topology
Network Technologies

- Ethernet
  - 1 Gbps and 10 Gbps common, 40 Gbps and 100 Gbps are available at higher cost

- Fiber Channel
  - 32-138 Gbps available

- Infiniband
  - a very-low-latency networking technology
    - 0.5 to 0.7 microseconds, compared to a few microseconds for optimized ethernet
Parallel Database Architectures

(a) shared memory

(b) shared disk

(c) shared nothing

(d) hierarchical
Shared Memory

- Processors (or processor cores) and disks have access to a common memory
  - Via a bus in earlier days, through an interconnection network today
- Extremely efficient communication between processors
- Downside: shared-memory architecture is not scalable beyond 64 to 128 processor cores
  - Memory interconnection network becomes a bottleneck

(a) shared memory
Modern Shared Memory Architecture
Cache Levels

- Cache line: typically 64 bytes in today's processors
- Cache levels within a single multi-core processor

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
<th>Core 2</th>
<th>Core 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 Cache</td>
<td>L1 Cache</td>
<td>L1 Cache</td>
<td>L1 Cache</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>L2 Cache</td>
<td>L2 Cache</td>
<td>L2 Cache</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Shared L3 Cache</td>
</tr>
</tbody>
</table>

- Shared memory system can have multiple processors, each with its own cache levels
Cache Coherency

- Cache coherency:
  - Local cache may have out of date value
  - Strong vs weak consistency models
  - With weak consistency, need special instructions to ensure cache is up to date

- Memory barrier instructions
  - **Store barrier (sfence)**
    - Instruction returns after forcing cached data to be written to memory and invalidations sent to all caches
  - **Load barrier (lfence)**
    - Returns after ensuring all pending cache invalidations are processed
  - **mfence instruction** does both of above

- Locking code usually takes care of barrier instructions
  - Lfence done after lock acquisition and sfence done before lock release
Shared Disk

- All processors can directly access all disks via an interconnection network, but the processors have private memories.
  - Architecture provides a degree of **fault-tolerance** — if a processor fails, the other processors can take over its tasks
    - the data of the failed processor is resident on disks that are accessible from all processors.
- Downside: bottleneck now occurs at interconnection to the disk subsystem.
Modern Shared Disk Architectures: via Storage Area Network (SAN)
Shared Nothing

- Node consists of a processor, memory, and one or more disks
- All communication via interconnection network
- Can be scaled up to thousands of processors without interference.
- Main drawback: cost of communication and non-local disk access; sending data involves software interaction at both ends.
Hierarchical

- Combines characteristics of shared-memory, shared-disk, and shared-nothing architectures.
  - Top level is a shared-nothing architecture
    - With each node of the system being a shared-memory system
  - Alternatively, top level could be a shared-disk system
    - With each node of the system being a shared-memory system

(d) hierarchical
Shared-Memory Vs Shared-Nothing

- Shared-memory internally looks like shared-nothing!
  - Each processor has direct access to its own memory, and indirect (hardware level) access to rest of memory
  - Also called non-uniform memory architecture (NUMA)

- Shared-nothing can be made to look like shared memory
  - Reduce the complexity of programming such systems by distributed virtual-memory abstraction
  - Remote Direct Memory Access (RDMA) provides very low-latency shared memory abstraction on shared-nothing systems
    - Often implemented on top of infiniband due it its very-low-latency
  - But careless programming can lead to performance issues
Distributed Systems

- Data spread over multiple machines (also referred to as sites or nodes).
- Local-area networks (LANs)
- Wide-area networks (WANs)
  - Higher latency

![Diagram showing communication via network between sites A, B, and C]
Distributed Databases

- **Homogeneous distributed databases**
  - Same software/schema on all sites, data may be partitioned among sites
  - Goal: provide a view of a single database, hiding details of distribution

- **Heterogeneous distributed databases**
  - Different software/schema on different sites
  - Goal: integrate existing databases to provide useful functionality

- Differentiate between **local transactions** and **global transactions**
  - A **local transaction** accesses data in the *single* site at which the transaction was initiated.
  - A **global transaction** either accesses data in a site different from the one at which the transaction was initiated or accesses data in several different sites.
Data Integration and Distributed Databases

- Data integration between multiple distributed databases

Benefits:

- Sharing data – users at one site able to access the data residing at some other sites.
- Autonomy – each site is able to retain a degree of control over data stored locally.
Availability

- **Network partitioning**
- **Availability** of system
  - If all nodes are required for system to function, failure of even one node stops system functioning.
  - Higher system availability through redundancy
    - data can be replicated at remote sites, and system can function even if a site fails.
Implementation Issues for Distributed Databases

- Atomicity needed even for transactions that update data at multiple sites

- The **two-phase commit protocol (2PC)** is used to ensure atomicity
  - Basic idea: each site executes transaction until just before commit, and the leaves final decision to a coordinator
  - Each site must follow decision of coordinator, even if there is a failure while waiting for coordinators decision

- 2PC is not always appropriate: other transaction models based on persistent messaging, and workflows, are also used

- Distributed concurrency control (and deadlock detection) required

- Data items may be replicated to improve data availability

- Details of all above in Chapter 24
Cloud Based Services

- Cloud computing widely adopted today
  - On-demand provisioning and **elasticity**
    - ability to scale up at short notice and to release of unused resources for use by others
- Infrastructure as a service
  - Virtual machines/real machines
- Platform as a service
  - Storage, databases, application server
- Software as a service
  - Enterprise applications, emails, shared documents, etc.
- Potential drawbacks
  - Security
  - Network bandwidth
Cloud Service Models

Cloud Clients
Web browsers, mobile apps, ...

internet

Software-as-a-Service
Enterprise applications,
email, shared documents, ...

Platform-as-a-Service
Data storage, Database,
Application server, ...

Infrastructure-as-a-Service
Virtual Machines, servers,
storage, ...
Application Deployment Alternatives

a) Multiple applications on a single machine

App
App
App
App
Libraries
OS Kernel

b) Each application running on its own VM, with multiple VMs running in a machine

App
Libraries
OS Kernel
App
Libraries
OS Kernel
Hypervisor

Individual Machines
Virtual Machines
(e.g. VMWare, KVM, ..)

c) Each application running in its own container, with multiple containers running in a machine

App
Libraries
App
Libraries
App
Libraries
OS Kernel

Containers
(e.g. Docker)
Application Deployment Architectures

- **Services**
- **Microservice Architecture**
  - Application uses a variety of services
  - Service can add or remove instances as required
- **Kubernetes supports containers, and microservices**
End of Chapter 20
Figure 17.11

SAN

storage array

storage array

tape library

LAN/WAN

server

server

client

client

client

data-processing center

Web content provider