Database Architectures

COSC 404 – Database System Implementation

Databases Architectures
Not "One Size Fits All"

Relational databases are still the dominant database architecture and apply to many data management problems.

- Over $30 billion annual market.

However, recent research and commercial systems have demonstrated that "one size fits all" is not true. There are better architectures for classes of data management problems:

- Transactional systems: In-memory architectures
- Data warehousing: Column stores, parallel query processing
- Big Data: Massive scale-out with fault tolerance
- "NoSQL": simplified query languages/structures for high performance, consistency relaxation

Variety of Database Architectures

A database system provides independence from data storage and processing challenges. There are many architectures that are good for different use cases.

- Single (centralized) server database – easy to deploy/use
- Parallel database – for large query loads and data sizes
- Distributed database – for large-scale deployments (shared-nothing) with physical/geographical distribution
- Virtual (multi-)database – for integrating existing, autonomous databases
- Data warehouses – for decision support queries
- NoSQL databases – MongoDB, Cassandra, etc. supporting different data models

There are also lots of ways for implementing these architectures with associated algorithms.

Single (Centralized) Server Database

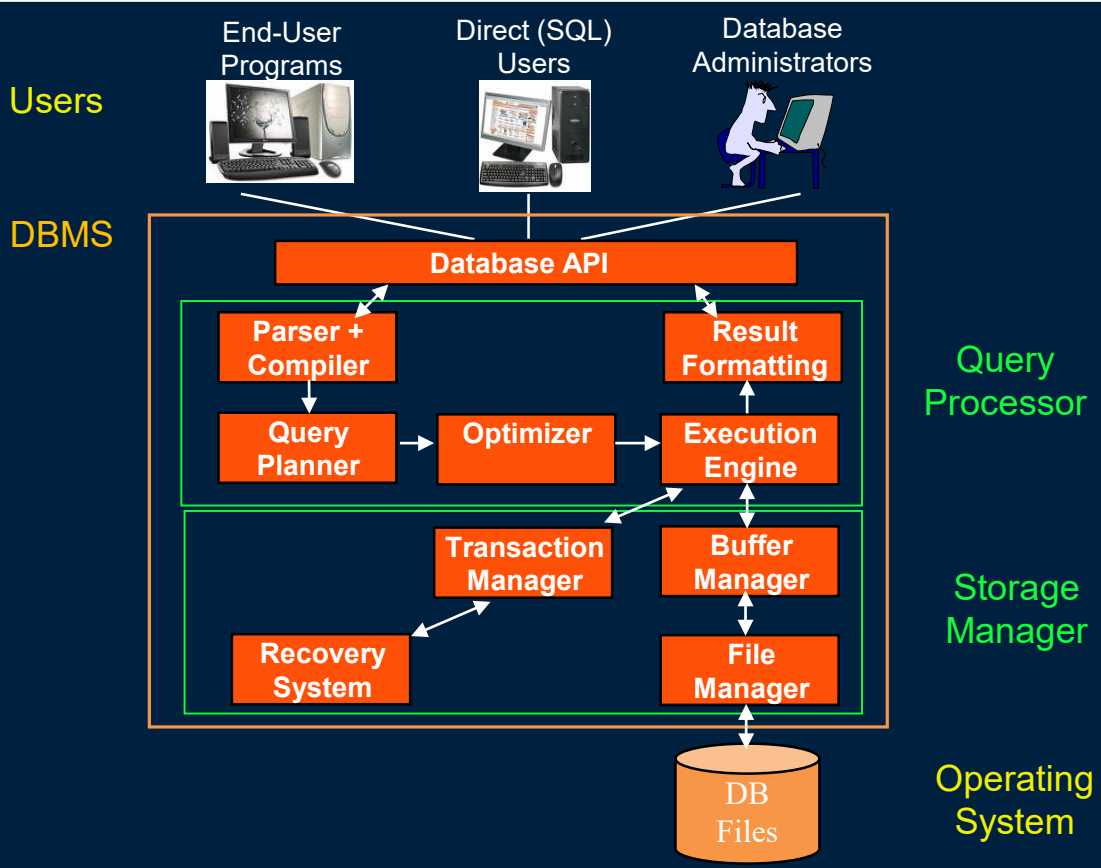
Single server centralized database systems such as MySQL, PostgreSQL, Oracle, and SQL Server have fairly standardized features and properties.

Ideal for: General-purpose databases (low cost/complexity)

Implementation details we studied:

- Data storage system, buffer manager
- Indexing algorithms and using indexes in practice
- Query processing/optimization of SQL
- Transactions, concurrency control, recovery
- Many systems also support distribution/replication/partitioning.
- Often, no parallelism within a query but can execute many queries simultaneously.
- Using JDBC API including PreparedStatements

DBMS Architecture



Parallel Database Systems

A ***parallel database system*** consists of multiple processors and storage connected by a fast interconnection network.

Ideal for: processing time-consuming decision-support queries or providing high throughput for transaction processing within a single server/data center

Implementation details:

- replication and partitioning used for availability/performance
- parallel algorithms for relational operators
- modified algorithms for concurrency control and transactions
- query optimization must consider data location

Parallel Database Systems
Greenplum

Greenplum is a shared-nothing, massively parallel (MPP) system where each node runs PostgreSQL.

Implementation:

- Cost-based optimizer factors in cost of moving data across nodes.
- Join and sort algorithms implemented in parallel across nodes and can move data between them.
- Utilizes log shipping and segment-level replication for fail-over.
- Supports SQL and Map-Reduce.
- Developed by Pivotal software (currently part of VMware).

Distributed Database System

A ***distributed database system*** is a database system distributed across several network nodes that appears to the user as a single system.

Ideal for: high availability/reliability where large data set can be partitioned and queried across servers (often geographically)

Implementation details:

- Shared-nothing, massively parallel (MPP) architectures
- Concurrency control must determine how to handle replication and partitioning (eager versus lazy consistency)
- Scaling requires dividing workload across servers and intelligent data placement and query processing

Primary/Secondary Replication

Primary/Secondary replication is supported by all major relational database systems (MySQL, PostgreSQL, Oracle, etc.).

Implementation details:

- 1) How are updates sent to secondary nodes? Log shipping or real-time.
- 2) Secondary nodes can except read requests but need to indicate when a transaction is read-only.
- 3) Secondary nodes can take over from primary if it fails.

Primary/Primary Replication

Primary/primary replication allows the data to be modified at more than one server. This requires coordination by the primary servers.

Techniques:

- Any update must be "approved" by all (or a majority) of the primary servers. This approval may be done before commit (online) using a distributed algorithm (e.g. two phase commit).
- Updates may be allowed on multiple servers simultaneously, but there must be some system or user-configured resolution mechanism to handle conflicts.

Oracle

Oracle supports multiple servers with distributed features:

- database links between databases for querying other databases as if the data was local to Oracle (virtualization)
- supports remote/distributed transactions that involve one or more nodes (via database links and 2PC)
- does not perform auto-fragmentation/location transparency but does support user configurable horizontal partitioning
- Oracle supports both primary/secondary and primary/primary replication using either synchronous or asynchronous propagation of changes between servers.
  - Different techniques for conflict resolution that user can control.
- Parallel execution of single SQL statement (joins, scans, sorts)

Oracle Real Application Clusters

Oracle Real Application Clusters (RAC) is a shared-storage architecture with multiple server nodes.

Provides support for clustering and high availability with multiple servers having concurrent access to the database and any server can process a transaction.

SQL Server

Microsoft SQL Server supports different use cases within its product including warehousing and in-memory databases.

- In-memory tables and query processing for transactional
- Data warehousing extensions and algorithms for analytics
- Replication using primary/secondary and primary/primary via log shipping, publish/subscribe, and merge conflict resolution
- Linked servers (ODBC) for heterogeneous query processing and virtualization
- Ability to scale from single server to multiple servers with high availability
- Most "reasonably-priced" of the commercial systems
- Very active database research laboratory

Database Architectures: NoSQL vs Relational

"NoSQL" databases are useful for several problems not well-suited for relational databases with some typical features:

- **Variable data:** semi-structured, evolving, or has no schema
- **Massive data:** terabytes or petabytes of data from new applications (web analysis, sensors, social graphs)
- **Parallelism:** large data requires architectures to handle massive parallelism, scalability, and reliability
- **Simpler queries:** may not need full SQL expressiveness
- **Relaxed consistency:** more tolerant of errors, delays, or inconsistent results ("eventual consistency")
- **Easier/cheaper:** less initial cost to get started

NoSQL is not really about SQL but instead developing data management architectures designed for scale.

- NoSQL – "Not Only SQL"

Data Warehouse Architectures

A ***data warehouse*** is a historical database that summarizes, integrates, and organizes data from one or more operational databases in a format that is more efficient for analytical queries.

Ideal for: Large-scale analytic and decision-support queries

Implementation details:

- Special storage formats (compressed, column stores)
- Special index structures (bitmap indexes)
- Optimized for reads over writes
- Large query rather than large number of queries/updates so parallelism within a query is critical
- May be relational or multidimensional (cubes).

In-Memory Databases

An ***in-memory database*** stores its working set of data in memory for improved response time.

Ideal for: high-volume, low-latency transactional systems

Implementation details:

- May be single or multiple server
- Data must be in memory. Persistent store used only in failure. Specialized memory queries (often have user pre-declare queries/transactions) – VoltDB, SQL Server, SAP HANA
- Concurrency control and recovery system optimized for high throughput and unlikely failures

Batch Systems
Map-Reduce

Batch systems like ***Map-Reduce*** designed for processing large-scale queries where the data may not be well-structured or pre-processed into a database engine.

Implementation Details:

- Data often has limited structure (flat files, log files, CSV).
  - Massive amounts of data that may not be worth loading into a database.
- Queries may take a LONG time so query processor must be resistant to failures with the ability to restart parts of the query that failed.
- Many database vendors have ability to integrate with Hadoop File System and perform Map-Reduce queries.

Cloud Databases

Cloud databases are databases hosted by a service provider that allow for easy setup, administration and scaling.

- Database as a service – databases hosted by provider, provide monitoring, backup, fail-over, high-availability, and ability to scale.

Examples: Google BigTable, Amazon RDS, DynamoDB, Redshift

Ideal for: Quick start without a server, minimal administration, scaling without expertise

Multi-Tenancy

Multi-tenancy is the ability to handle multiple customers (tenants) on the same database infrastructure. Approaches:

- **Separate server** – each tenant has their own physical hardware, OS, DBMS
- **Shared server, separate DBMS** – shared hardware but have multiple different DBMS running on hardware (maybe VMs)
- **Shared database server, separate databases** – shared DBMS but different databases
- **Shared database, separate schema** – same database but multiple schemas (user collection of objects)
- **Shared database, shared schema** – customer data is differentiated by tenant id in all tables designed

Multi-Tenancy Issues

Multi-tenancy issues to consider:

- Hardware and software costs
- Efficient use of hardware resources
- Isolation and security
- Query performance
- Ease of backup

Bottom Line

Bottom line: **No one size fits all.**

Select a database system based on your application and use case.

Understanding how database systems work and their architectures will help you make informed decisions on database systems to use and how to deploy them properly.

Survey Question: Lecture Value

Question: On a scale of 1 to 5 with 5 being the highest, how valuable/useful was the lecture time?

A) 1

B) 2

C) 3

D) 4

E) 5

Survey Question:
Lab Assignment Value

Question: On a scale of 1 to 5 with 5 being the highest, how valuable/useful were the lab assignments?

A) 1

B) 2

C) 3

D) 4

E) 5

Survey Question: Workload

Question: On a scale of 1 to 5 with 1 being very low and 5 being very high, how was the overall workload compared to other courses and your expectations?

A) 1

B) 2

C) 3

D) 4

E) 5

Survey Question: Clicker Value

Question: On a scale of 1 to 5 with 5 being the highest, how valuable/useful were the clicker questions used in-class?

A) 1

B) 2

C) 3

D) 4

E) 5

Summary of Course

Our course goals were to understand database systems to:

1) Be a better, "expert" user of database systems.

2) Be able to use and compare different database systems.

3) Adapt the techniques when developing your own software.

We opened the database system "**black box**".

- Inside was storage, indexing, query processing/optimization, transactions, concurrency, recovery, distribution, lots of stuff!

You gained ***lots*** of industrial experience using a variety of databases and became a better, more experienced developer.

- MySQL, PostgreSQL, Microsoft SQL Server, MongoDB, JUnit, Snowflake, Java, JDBC, javacc, JSON, Map-Reduce, SQL

Thank you for a great course!

Good luck on the exam!


UBC
THE UNIVERSITY OF BRITISH COLUMBIA