Database System Implementation

Joy Arulraj

Slides are derived from courses developed by Thomas Neumann and Andy Pavlo.

Course Overview

Welcome!

- This course is on the design and implementation of database management systems (DBMSs).

Why you might want to take this course?

- DBMS developers are in demand.
- There are many challenging unsolved problems in data management and processing.
- If you are good enough to write code for a DBMS, then you can write code on almost anything else.

Why you might **not** want to take this course?

- This is not a course on how to use a database to build applications or how to administer a database.

Course Objectives

- Learn about modern practices in database internals and systems programming.
- Students will become proficient in:
  - Writing correct + performant code
  - Proper documentation + testing
  - Working on a large systems programming project

Course Topics

The internals of single node systems for disk-oriented and in-memory databases.

Topics include:

- Relational Databases
- Storage
- Indexing
- Query Execution
- Potpourri

Background

- Assume that you have taken an introductory course on database systems (*e.g.*, GT 4400).
- All programming assignments will be written in C++17.
  - Be prepared to develop and test a multi-threaded program.
  - Assignment 1 will help get you caught up with C++.
  - If you have not encountered C++ before and are a Java programmer, you will need to pick C++ yourself.
  - Here a couple of helpful references: ❶ Java to C++ Transition Tutorial, ❷ C++ Language
  - I will briefly cover relevant parts of C++ in this course.

Course Logistics

- Course Policies
  - The programming assignments and exercise sheets must be your own work.
  - They are **not** group assignments.
  - You may **not** copy source code from other people or the web.
  - Plagiarism will **not** be tolerated.
- Academic Honesty
  - Refer to Georgia Tech Academic Honor Code.
  - If you are not sure, ask me.

Course Logistics

- Course Web Page
  - Schedule: https://www.cc.gatech.edu/ jarulraj/courses/4420-f20/
- Discussion Tool: Piazza
  - https://piazza.com/configure-classes/fall2020/cs44206422
  - For all technical questions, please use Piazza
  - Don't email me directly
  - All non-technical questions should be sent to me
- Grading Tool: Gradescope
  - You will get immediate feedback on your assignment.
  - You can iteratively improve your score over time.
- Virtual Office Hours
  - Will be posted on Piazza.

Course Rubric

- Programming Assignments (**55**%)
  - Five assignments based on the BuzzDB academic DBMS.
  - Each assignment builds on the previous one.
- Exercise Sheets (**15**%)
  - Three pencil-and-paper tasks.
  - You will need to upload the sheets to Gradescope.
- Exams (**30**%)
  - Two remote exams.
  - We are planning to use the online proctoring service provided by the university.

Late Policy

- You are allowed <u>**four**</u> slip days for either programming assignments or exercise sheets.
- You lose 25% of an assignment's points for every 24 hrs it is late.
- Mark on your submission (1) how many days you are late and (2) how many late days you have left.

Teaching Assistants

- Gaurav Tarlok Kakkar
  - M.S. (Computer Science)
  - Worked at Adobe (2 years).
  - Research Topic: Video analytics using deep learning.
- Pramod Chundhuri
  - Ph.D. (Computer Science)
  - Research Topic: Video analytics using deep learning.
- If you are acing through the assignments, you might want to hack on the video analytics system (codenamed EVA) that we are building.
- Drop me a note if you are interested!

Motivation

Motivation (1)

A **Database Management System** (DBMS) is a software that allows applications to store and analyze information in a database.

A general-purpose DBMS is designed to allow the definition, creation, querying, update, and administration of databases.

DBMSs are super important

- core component of most computer applications
- very large data sets
- valuable data

Motivation (2)

Key challenges:

- scalability to huge data sets
- reliability
- concurrency

Results in very complex software.

About This Course

Goals of this course

- learning how to build a modern DBMS
- understanding the internals of existing DBMSs
- understanding the impact of hardware properties

Rough structure of the course

1. Relational Databases
2. Storage
3. Indexing
4. Query Execution

Next Course

In a follow-up course offered in the Spring semester (GT 8803), we will focus on:

1. Query Compilation
2. Concurrency Control
3. Recovery
4. Query Optimization
5. Potpourri

This course will be a pre-requisite for the next course.

Textbook

- Silberschatz, Korth, & Sudarshan: *Database System Concepts*. McGraw Hill, 2020.
- Hector Garcia-Molina, Jeff Ullman, and Jennifer Widom: *Database Systems: The Complete Book*. Prentice-Hall, 2008.

Caveat

- These textbooks mostly focus on traditional disk-oriented database systems
- Not modern in-memory database systems

Motivational Example

Why is a DBMS different from most other programs?

- many difficult requirements (reliability etc.)
- but a key challenge is **scalability**

Motivational example

Given two lists L_1 *and* L_2*, find all entries that occur on both lists.*

Looks simple...

$L_1 = \{1, 2, 3, 5\}$

$L_2 = \{1, 5, 3, 4, 7\}$

$L_1 \cap L_2 = \{1, 3, 5\}$

Motivational Example (2)

Given two lists L_1 *and* L_2*, find all entries that occur on both lists.*

Simple if both fit in main memory
Don't need more than a few lines of code

Motivational Example (2)

Given two lists L_1 *and* L_2*, find all entries that occur on both lists.*

Simple if both fit in main memory
Don't need more than a few lines of code

- sort both lists and intersect $L_1 = \{1, 2, 3, 5\}$; $L_2 = \{1, 3, 4, 5, 7\}$
- or load one list in an **unordered hash table** [2] and probe
- or load one list in an **ordered tree** structure [1]
- or ...

Note: pairwise comparison is not an option! $O(n^2)$
We will discuss about hash tables and B+trees in ▸ Access Paths.

Motivational Example (3)

Given two lists L_1 *and* L_2*, find all entries that occur on both lists.*

Slightly more complex if **only one list** fits in main memory

Motivational Example (3)

Given two lists L_1 *and* L_2*, find all entries that occur on both lists.*

Slightly more complex if **only one list** fits in main memory

- load the smaller list into memory
- build tree structure/sort/hash table/...
- scan the larger list one **chunk** (*e.g.*, 10 numbers) at a time
- search for matches in main memory

Code still similar to the pure main-memory case.

Motivational Example (4)

Given two lists L_1 *and* L_2*, find all entries that occur on both lists.*

Difficult if neither list fits into main memory

Motivational Example (4)

Given two lists L_1 *and* L_2*, find all entries that occur on both lists.*

Difficult if neither list fits into main memory

- no direct interaction possible
- Option 1: Sorting works, but already a difficult problem
  - Programming Assignment 1: **external** merge sort
  - We will cover this in ▸ External Hash Join.
- Option 2: Partitioning scheme (*e.g.*, numbers in [1, 100], [101, 200],. . .)
  - break the problem into smaller problems
  - ensure that each partition fits in memory

Code significantly more involved.

Motivational Example (5)

Given two lists L_1 *and* L_2*, find all entries that occur on both lists.*

Hard if we make no assumptions about L_1 and L_2.

Motivational Example (5)

Given two lists L_1 *and* L_2*, find all entries that occur on both lists.*

Hard if we make no assumptions about L_1 and L_2.

- tons of corner cases
- a list can contain duplicates
- a single duplicate value might exceed the size of main memory!
- breaks "simple" external memory logic
- multiple ways to solve this
- but all of them are somewhat involved
- and a DBMS must not make assumptions about its data!

Code complexity is very high.

Motivational Example (6)

Designing a robust, scalable algorithm is hard

- must cope with very large instances
- hard even when the database fits in main memory
- billions of data items
- rules out the possibility of using $O(n^2)$ algorithms
- external algorithms (*i.e.*, database does not fit in memory) are even harder

This is why a DBMS is a complex software system.

Shift in Hardware Trends

Traditional Assumptions

Historically, a DBMS is designed based on these assumptions:

- database is much larger than main memory
- I/O cost dominates everything with **Hard Disk Drives** (HDD)
- random I/O operations to "mechanical" HDD are very expensive

This led to a very **conservative**, but also very **scalable** design.

Hardware Trends

Hardware has evolved over the decades (invalidating these assumptions):

- main memory size is increasing
- servers with 1 TB main memory are affordable
- "electromagnetic" **Solid State Drives** (SSD) have lower random I/O cost
- ...

This affects the design of a DBMS

- CPU costs are now more important
- I/O operations are eliminated or greatly reduced
- the classical architecture (**disk-oriented database systems**) has become suboptimal

But this is more of an evolution as opposed to a revolution. Many of the old techniques are still relevant for scalability.

Goals

Ideally, a DBMS

- efficiently handles arbitrarily-large databases
- never loses data
- offers a high-level API to manipulate and retrieve data
- this API is the **declarative Structured Query Language** (SQL)
- shields the application from the complexity of data management
- offers excellent performance for all kinds of queries and all kinds of data

This is a very ambitious goal!
This has been accomplished, but comes with inherent complexity.

Course Organization

1. storage
2. access paths
3. query processing (algebraic operators)

In each topic, we will cover aspects of both disk-oriented and modern in-memory DBMSs.

Conclusion

- Complexity of a database system arises from the need for robust, scalable algorithms
- A database systems must satisfy many requirements: reliability, scalability, *e.t.c*.
- In the next lecture, we will learn about relational database systems.

References I

[1] CPPReference. std::map. https://en.cppreference.com/w/cpp/container/map.

[2] CPPReference. std::unordered_map. https://en.cppreference.com/w/cpp/container/unordered_map.