Database Systems on Modern CPU Architectures

Ilaria Battiston [*]

Summer Semester 2020-2021

[*] All notes are collected with the aid of material provided by T. Neumann. All images have been retrieved by slides present on the TUM Course Webpage.

Contents

1 Introduction

Database systems are extremely important in the real world, in all use cases involving management of large quantities of data: of course, they need to provide some guarantees such as scalability, reliability and concurrency, which can be a challenge.

There are scenarios which can either be optimized, or generate some terrible runtime, such as a simple join: its complexity can vary from $O(n)$ (hash join) to $O(n^2)$ with a nested loop. Obviously, the latter case is ruled out since it forbids any kind of scalability.

This can get even more complex when one of the data sources fits in memory, and the other does not: in this case, an additional index structure is employed, to perform lookups in external memory.

When neither fits, sorting is an option, although already a difficult problem; another approach is partitioning, breaking data into sub-problems until they fit in memory.

In general, there are many corner cases to consider, and sometimes it is necessary to make assumptions about the data, making code complexity very high.

Historically, DMBS are designed such that I/O operations are random and expensive, and data is much larger than main memory. Conservative designs are scalable, yet in modern hardware systems main memory size is increasing, reducing I/O costs.

This has consequences for database implementation, since the old architecture becomes suboptimal. Ideally, DBMS should be adapted in order to also maintain durability and good performance.

1.1 Set-oriented processing

Set-oriented processing is a way to avoid using nested loops (accessing every element of one list and then each element of the other) and subsequent squared runtimes, as this operation is not optimal.

It is helpful to perform operations for large batches of data, handling input in sets. Notation takes advantage of relational algebra operations, since these are already set-oriented; even in the case of duplicates (bags), those can still be mapped to unique values.

Algebra needs some extensions for real-world scenarios, such as map, group by or dependent join (predicates which must be evaluated after others).

In short, processing whole batches of tuples can help sorting, hashing, re-organizing the data and creating index structures, ideally achieving worst-case logarithmic runtime.