CMPSCI 630 Systems **Spring 2017**

Lecture 18

Lecturer: Emery Berger *Scribe: Andrew Danise, Arjun Sreedharan*

18.1 Databases

18.1.1 History

Early Databases were very primitive and did not have simple querying capabilities. CODASYL exposes an API for general purpose programming languages. Lookups were done using a minimal abstraction built using iterators via a cursor. It maintained an index made up of a bunch of pointers to data records.

The development of SQL (Structured Query Language) started out at IBM. It was preceded by a language named Quel which was based on tuples. IBM's Database research in the early stages followed 2 approaches to building querying capability:

- **SQL**
- **QBE** (Query By Example): QBE is a visual way of expressing queries. The graphical queries would look similar to ASCII art. QBE is very easy for untrained users to understand, but it is difficult to express complex queries.

18.1.2 Structure

Databases use B-Tree or B+ Tree data structures. B-tree is a generalization of a Binary Search Tree where each node can have multiple children. B-trees are used to implement database indices. They maximize locality when searching for a particular key. B-trees are very efficient when represented on disk. Size of a B-tree node is typically a 4K page.

Unlike B-tree, B+ trees don't have data associated with interior nodes and more keys fit on a page of memory.

18.1.3 Partitioning

18.1.3.1 Range Partitioning

In Range partitioning, data is partitions based on the range of values of a particular key. For example, if *Name* is the partitioning key, all entries will with name beginning in letter from A to J can be in one partition, those beginning with K to P in another and the rest in the third. This gives a good speedup if the queries can be parallely executed in the different partitions. The size of partitions may not be equal.

18.1.4 Hash Partition

Hash Partitioning is based on a hashing algorithm that is applied to the partitioning key. The hashing function evenly spreads the entries among partitions, giving partitions approximately the same size. It is ideal for balancing data storage evenly across machines or replicas.

There are also other methods for partitioning like Horizontal and Vertical partitioning (sharding). In Horizontal partitioning, entries in the database are divided into different partitions. In Vertical Partitioning, the fields of the entries are divided into different partitions.

18.2 Query Optimization

Sometimes the order in which the query is executed has a drastic impact on the performance of the query. In addition, sometimes the database can optimize queries because it knows about some inherent properties of the data. For example, if you run the the SQL query "select * from DB where salary ¿ 10000 and name == "John" at a company where everyone makes more than 10000 it will need to lookup all of the data. However, if the database reorganized the query to first check if the name == John than the results would be more limited. In order to perform such optimizations, databases store statistics on the type of data they hold. They use this data to reorganize queries to achieve the best possible performance. This is called Selectivity.

18.3 MapReduce

Databases are great for storing structured data. But they do not scale well when unstructured on semi-structured data is stored. Assume that an organization has to store many HTML or XML documents. The overhead of query optimizer and other management overhead for databases makes its scalability very limited. This is where a restricted model of distributed computing named MapReduce comes into the picture. MapReduce is highly scalable with unstructured documents. MapReduce works on the basis of *map* and *reduce* methods provided by the user. It provides in-built fault tolerance, and at every stage writes data to disk.