COMPSCI 590S Systems for Data Science **Fall 2016**

Lecture 8:

Lecturer: Emery Berger *Scribe(s): Boya Ren, Andrew Bass*

8.1 Parallel Databases

The motivation for parallel databases is two folds. First, it takes advantage of multiple processes. In the ideal case, if we have 100 computers, we want our database operations to be 100x faster. Second, we want to use multiple disks, such that 100 computers gives us 100X storage space.

Suppose in a database we have several files as listed in the figure below. If we want to store each file on a machine, it does not work because the file might be too big to fit in. As a result, we need strategies to partition to files.

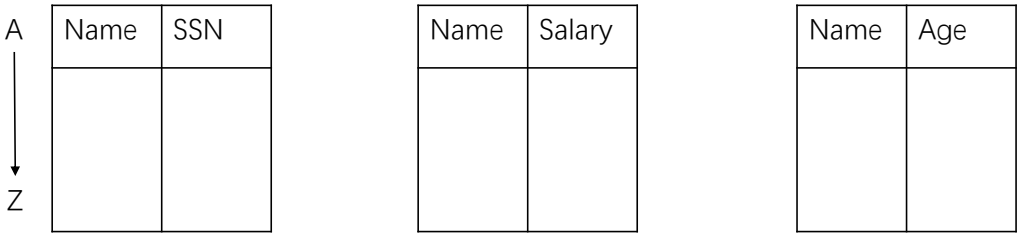

Figure 8.1: An Example of Database Files

8.1.1 Database Partitioning

A naive way to partition the database is to horizontally divide it into several parts in the alphabetical order. For example, as illustrated below, data entries with name A to C is stored in one machine, D to F in the second and so on.

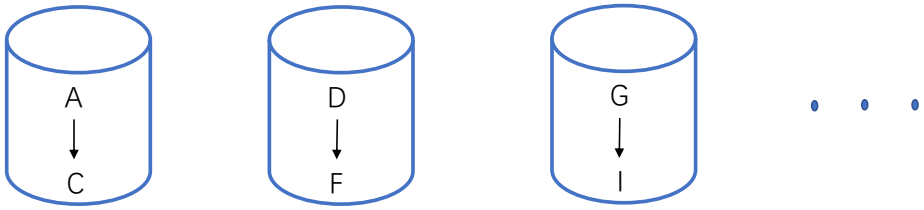

Figure 8.2: Horizontal Partitioning

Problem of this scenario is obvious: the data is not divided evenly, with much more items in some machines and less in others. We want our partitioning scheme to achieve better **load balance**, i.e. data spreads out

evenly. The simplest solution is to distribute the data evenly into different sub-databases. For example, the first 1000 lines in the first server, the next 1000 in the second, etc. But this makes query hard. If we want to request the data for "John Franklin", we have no idea where it is really stored. Additionally, as the data set grows, we would need to add more machines, and would not be able to distribute the new data evenly among the current set of machines.

One way to address the partition function is to use hash function. It maps the input x to ith server by $h(x) = i$. Suppose we have N machines, a good hash function will make the output "look random", i.e. $p[h(x) = i] \approx 1/N$ is close to uniform distribution. The key used as the input to the hash function could be anything, a unique id (such as the users SSN) or a combination of columns. This approach also allows easy look-up of which machine the data is stored on, simply by hashing the same key used to store.

8.1.2 Distributed Database v.s. MapReduce

For a distributed system, our goal is to maximize local computation, parallelism, RAM utilization, and minimize communication overhead.

The biggest difference between database and MapReduce is that database has schemas but MapReduce is scheme-less. For example, a social record database usually defines variables like "Name" String type of length 30, SSN string type length 9, Salary float type and so on. But MapReduce always deals with unstructured data like "XML"/HTML, webpage which is "structured" hierachical, URL and so on. These can be variable sized and may contain unusual structures and hierarchies.

Modern databases has introduced Binary Large Objects (BLOBs), which are variable sized and harsh partitioned. It partly breaks the schema requirement of database, but still not as flexible as MapReduce.

There are generally two query plans, one is "distribute collect" and the other is "issue", since some queries have more computation, while some have more communication.

8.1.3 Row-Based DB v.s. Column-based DB

As illustrated in the figure below, row-based database stores one row after another, each row corresponding to a whole item. Column-based database, however, take out all the columns, each stored independently.

The advantages of row-based DB is obvious: if we want to append or delete an entry, it only takes one operation. In the column-based DB, however, both take n operation, since we have to change every column.

However, for column-based db, since there will be some frequent terms in each column, we can use compression methods such as Huffman Coding (more frequent terms will be coded with less bits) or Run Length Encoding (.tiff).

8.1.4 Transactions

From the data analytics point of view, some databases are read only or read mostly, like ETL (Extract, transform, load). However, some databases are update-heavy, and thus database management systems (DBMS) are required. Usually, there is a priority based scheme (queue) to access resources. However, in distributed DB systems, some queries are processed simultaneously, and consequently, there will be conflicts and unsychronizations. That is why transactions are defined and used.

The word "transaction" is borrowed from financial transaction, which is supposed to be distributed, efficient, and highly fault-tolerant. Thus, a transaction satisfies ACID properties:

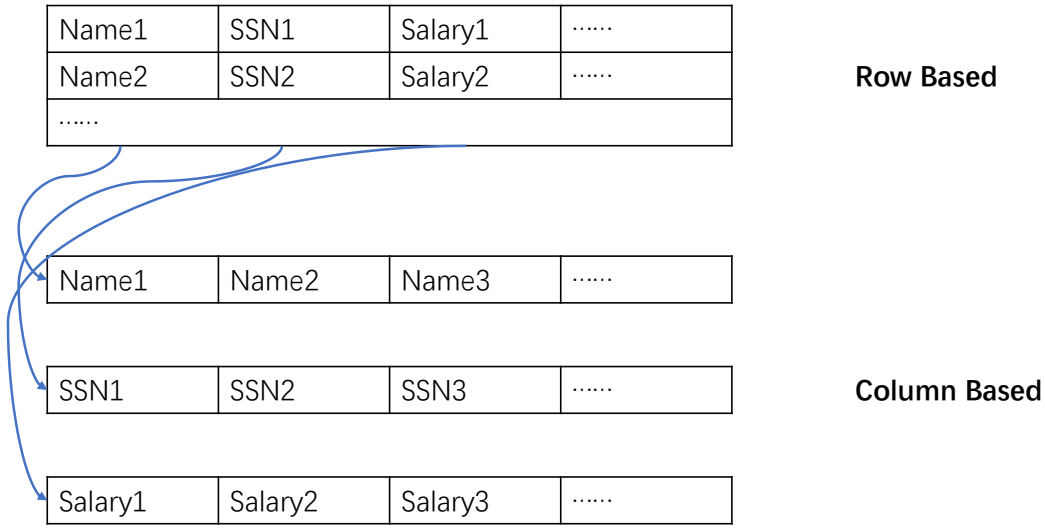

Figure 8.3: Row-based and Column-based

- Atomic: all or nothing. Either all operations in the transaction are done, or nothing appears.
- Consistent: a transaction transfers the system from one good state to another.
- Isolated: one transaction cannot see updates from other transactions.
- Durable: transactions are saved to some persistent stores.

What type of transactions are required is domain specific and ACID may be more strict than necessary. For example to keep track of Facebook 'like' counts, is sufficient to achieve 'eventual consistency', such that users may not see updates to the count immediately.

In order to enforce ACID transactions, two techniques will be applied.

- Locking
- Optimizations for transactions (lock-free)

8.1.5 NoSQL

NoSQL is a kind of database that implements distributed hash tables. Data is stored in key-value stores. For example, put(k,v) will store (k,v) pair, and get(k) fetches value v.