Lecture #14: Parallel Execution

15-445/645 Database Systems (Fall 2018)
https://15445.courses.cs.cmu.edu/fall2018/
Carnegie Mellon University
Prof. Andy Pavlo

1 Background

All the DBMS to execute queries in parallel provides several benefits:

- Increased performance in throughput and latency.
- Increased availability.
- Potentially lower *total cost of ownership* (TCO).

In parallel or distributed DBMS, the database is spread out across multiple resources to improve parallelism. The database "appears" as a single database instance to the application. The SQL query for a single-node DBMS should generate the same result on a parallel or distributed DBMS.

Parallel DBMS:

- Nodes are physically close to each other.
- Nodes are connected with high-speed LAN.
- Communication cost between nodes is assumed to be fast and reliable.

Distributed DBMS:

- Nodes can be far from each other.
- Nodes are connected using public network.
- Communication costs between nodes is slower and failures cannot be ignored.

Types of Parallelism

- **Inter-Query:** The DBMS executes different queries are concurrently. This increases throughput and reduces latency. Concurrency is tricky when queries are updating the database.
- **Intra-Query:** The DBMS executes the operations of a single query in parallel. This decreases latency for long-running queries.

2 Process Models

A DBMS *process model* defines how the system supports concurrent requests from a multi-user application/environment. The DBMS is comprised of more or more *workers* that are responsible for executing tasks on behalf of the client and returning the results.

Approach #1 – Process per Worker:

- Each worker is a separate OS process, and thus relies on OS scheduler.
- Use shared memory for global data structures.
- A process crash does not take down entire system.

Approach #2 – Process Pool:

- A worker uses any process that is free in a pool.
- Still relies on OS scheduler and shared memory.
- This approach can be bad for CPU cache locality due to no guarantee of using the same process between queries.

Approach #3 – Thread per Worker:

- Single process with multiple worker threads.
- DBMS has to manage its own scheduling.
- May or may not use a dispatcher thread.
- Although a thread crash (may) kill the entire system, we have to make sure that we write high-quality code to ensure that this does not happen.

Using a multi-threaded architecture has advantages that there is less overhead per context switch and you do not have to manage shared model. The thread per worker model does not mean that you have intra-query parallelism.

For each query plan, the DBMS has to decide where, when, and how to execute:

- How many tasks should it use?
- How many CPU cores should it use?
- What CPU core should the tasks execute on?
- Where should a task store its output?

3 Inter-Query Parallelism

The goal of this type of parallelism is to improve the DBMS's overall performance by allowing multiple queries to execute simultaneously.

We will cover this in more detail when we discuss concurrency control protocols.

4 Intra-Query Parallelism

The goal of this type of parallelism is to improve the performance of a single query by executing its operators in parallel. There are parallel algorithms for every relational operator.

Intra-Operator Parallelism

- The query plan's operators are decomposed into independent instances that perform the same function on different subsets of data.
- The DBMS inserts an **exchange** operator into the query plan to coalesce results from children operators. The exchange operator prevents the DBMS from executing operators above it in the plan until it receives all of the data from the children.

Inter-Operator Parallelism

- Operations are overlapped in order to pipeline data from one stage to the next without materialization. This is sometimes called **pipelined parallelism**.
- This approach is not widely used in traditional relation DBMSs. Not all operators can emit output until they have seen all of the tuples from their children. This is more common in *stream processing systems*, systems that continually execute a query over a stream of input tuples.

5 I/O Parallelism

Using additional processes/threads to execute queries in parallel will not improve performance if the disk is always the main bottleneck. Thus, we need a way to split the database up across multiple storage devices.

Multi-Disk Parallelism

Configure OS/hardware to store the DBMS's files across multiple storage devices. Can be done through storage appliances and RAID configuration. This is transparent to the DBMS. It cannot have workers operate on different devices because it is unaware of the underlying parallelism.

File-based Partitioning

Some DBMSs allow you to specify the disk location of each individual database. The buffer pool manager maps a page to a disk location. This is also easy to do at the file-system level if the DBMS stores each database in a separate directory. However, the log file might be shared.

Logical Partitioning

Split single logical table into disjoint physical segments that are stored/managed separately. Such partitioning is ideally transparent to the application. That is, the application should be able to access logical tables without caring how things are stored.

Vertical Partitioning:

- Store a table's attributes in a separate location (like a column store).
- Have to store tuple information to reconstruct the original record.

Horizontal Partitioning:

- Divide the tuples of a table into disjoint segments based on some partitioning keys.
- There are different ways to decide how to partition (e.g., hash, range, or predicate partitioning). The efficacy of each approach depends on the queries.