Lecture 14: Parallel Execution

15-445/645 Database Systems (Fall 2017)
Carnegie Mellon University
Prof. Andy Pavlo

Why parallel Execution is Important

1. Increased performance in throughput and latency
2. Increased availability
3. Potentially lower Total Cost of Ownership (TCO)

Parallel and Distributed Database Systems

1. In parallel or distributed systems, the database is spread out across multiple resources to improve parallelism
2. It's important that it 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
3. Parallel DBMSs
   (a) Nodes are physically close to each other
   (b) Nodes connected with high-speed LAN
   (c) **Communication cost is assumed to be small**
4. Distributed DBMSs
   (a) Nodes can be far from each other
   (b) Nodes connected using public network
   (c) **Communication cost and problems cannot be ignored**

Inter- vs Intra-Query Parallelism

1. **Inter-Query**: Different queries are executied concurrently. Increases throughput and reduces latency. Concurrency is tricky when queries are updating the database
2. **Intra-Query**: Execute the operations of a single query in parallel. Decreases latency for long-running queries

Process Models

1. A DBMS **process model** defines how the system is architected to support concurrent requesys from a multi-user application/environment

2. A **worker** is the DBMS component that is responsible for executing tasks on behalf of the client and returning the results

3. **Approach #1 – Process per Worker**

   (a) Each worker is a separate OS process, and thus relies on OS scheduler

   (b) Use shared memory for global data structures

   (c) A process crash doesn't take down entire system

4. **Approach #2 – Process Pool**

   (a) A worker uses any process that is free in a pool

   (b) Still relies on OS scheduler and shared memory

   (c) Bad for CPU cache locality due to no guarantee of using the same process between queries

5. **Approach #3 – Thread per Worker**

   (a) Single process with multiple worker threads

   (b) DBMS has to manage its own scheduling

   (c) May or may not use a distpatcher thread

   (d) Downside: a thread crash (may) kill the entire system

   (e) More modern processing model

6. Using a multithreaded architecture has advantages that there is less overhead per context switch and you dont have to manage shared model

7. The thread per worker model does not mean that you have intra-query parallelism

Worker Scheduling

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

   (a) How many tasks should it use?

   (b) How many CPU cores should it use?

   (c) What CPU core should the tasks execute on?

   (d) Where should a task store its output?

2. Reminder: The DBMS **always** knows more than the OS

Inter-Query Parallelism

1. Purpose: Improve overall performance by allowing multiple queries to execute simultaneously
2. Its important to provide the illusion of isolation through the **concurrency control** scheme, something that is extremely difficult

Intra-Query Parallelism

1. Purpose: Imporve the performance of a single query by executing its operators in parallel
2. Two Approaches
   (a) Intra-Operator
   (b) Inter-Operator
3. These two techniques are **not** mutually exclusive
4. There are parallel algorithms for every relational operator
5. **Intra-Operator Parallelism**
   (a) Operators are decomposed into independent instances that perform the same function on different subsets of data
   (b) The DBMS inserts an **exchange** operator into the query plan to coalesce results from children operators
6. **Inter-Operator Parallelism**
   (a) Operations are overlapped in order to pipeline data from one stage to the next without materialization
   (b) Also called **pipelined parallelism**
   (c) 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
   (d) This is more common in **stream processing systems**, systems that process a stream of input

I/O Parallelism

1. Using additional processes/threads to execute queries in parallel won't help if the disk is always the main bottleneck
2. Solution: Split the DBMS installation across multiple storage devices
3. **Multi-Disk parallelism**
   (a) Configure OS/hardware to store the DBMS's files across multiple storage devices
   (b) Can be done through storage appliances and RAID configuration

   (c) This is transparent to the DBMS

4. **Database partitioning**
   (a) Some DBMSs allow you to specify the disk location of each individual database
   (b) The buffer pool manager maps a page to a disk location
   (c) This is also easy to do at the filesystem level if the DBMS stores each database in a separate directory. However, the log file might be shared

5. **Partitioning**
   (a) Split single logical table into disjoint physical segments that are stored/managed separately
   (b) Ideally partitioning is transparent to the application. The application should be able to access logical tables without caring how things are stored
   (c) This is not always the case though
   (d) **Vertical Partitioning**
      i. Store a table's attributes in a separate location
      ii. Have to store tuple information to reconstruct the original record
   (e) **Horizontal Partitioning**
      i. Divide the tuples of a table into disjoint segments based on some partitioning keys
      ii. There's different ways to decide how to parition (e.g. hash, range, or predicate partitioning)

Conclusion

1. Parallel execution is important
2. Almost every DBMS support parallel execution
3. This is really hard to get right
   (a) Coordination overhead
   (b) Scheduling
   (c) Concurrency issues
   (d) Resource contention