CHAPTER 17

Database-System Architectures

The chapter is suitable for an introductory course. We recommend covering it, at least as self-study material, since students are quite likely to use the non-centralized (particularly client-server) database architectures when they enter the real world. The material in this chapter could potentially be supplemented by the two-phase commit protocol (2PC), (Section 19.4.1 from Chapter 19) to give students an overview of the most important details of non-centralized database architectures.

Exercises

17.7 Why is it relatively easy to port a database from a single processor machine to a multiprocessor machine if individual queries need not be parallelized?

Answer: Porting is relatively easy to a shared memory multiprocessor machine. Databases designed for single-processor machines already provide multitasking, allowing multiple processes to run on the same processor in a time-shared manner, giving a view to the user of multiple processes running in parallel. Thus, coarse-granularity parallel machines logically appear to be identical to single-processor machines, making the porting relatively easy.

The only difference is that a multiprocessor machine has multiple processor caches, and depending on the specific machine architecture, it is possible that a memory write executed on one processor may not be visible to a read on another processor for some time. To ensure that writes are visible at other processors, the database system needs to execute a special instruction (often called a *fence* instruction) which flushes a data item from all processor caches, ensuring the next read sees the latest value. Typically this instruction is executed just before releasing a latch or lock.

Porting a database to a shared disk or shared nothing multiprocessor architecture is clearly a little harder.

17.8 Transaction-server architectures are popular for client–server relational databases, where transactions are short. On the other hand, data-server

architectures are popular for client–server object-oriented database systems, where transactions are expected to be relatively long. Give two reasons why data servers may be popular for object-oriented databases but not for relational databases.

Answer: Data servers are good if data transfer is small with respect to computation, which is often the case in applications of OODBs such as computer aided design. In contrast, in typical relational database applications such as transaction processing, a transaction performs little computation but may touch several pages, which will result in a lot of data transfer with little benefit in a data server architecture. Another reason is that structures such as indices are heavily used in relational databases, and will become spots of contention in a data server architecture, requiring frequent data transfer. There are no such points of frequent contention in typical OODB applications which involve complex data that is rarely updated concurrently by multiple processes.

17.9 What is lock de-escalation, and under what conditions is it required? Why is it not required if the unit of data shipping is an item?

Answer: In a client-server system with page shipping, when a client requests an item, the server typically grants a lock not on the requested item, but on the *page* having the item, thus implicitly granting locks on all the items in the page. The other items in the page are said to be *prefetched*. If some other client subsequently requests one of the prefetched items, the server may ask the owner of the page lock to transfer back the lock on this item. If the page lock owner doesn't need this item, it de-escalates the page lock that it holds, to item locks on all the items that it is actually accessing, and then returns the locks on the unwanted items. The server can then grant the latter lock request.

If the unit of data shipping is an item, there are no coarser granularity locks; even if prefetching is used, it is typically implemented by granting individual locks on each of the prefetched items. Thus when the server asks for a return of a lock, there is no question of de-escalation, the requested lock is just returned if the client has no use for it.

17.10 Suppose you were in charge of the database operations of a company whose main job is to process transactions. Suppose the company is growing rapidly each year, and has outgrown its current computer system. When you are choosing a new parallel computer, what measure is most relevant—speedup, batch scaleup, or transaction scaleup? Why?

Answer: With increasing scale of operations, we expect that the number of transactions submitted per unit time increases. On the other hand, we wouldn't expect most of the individual transactions to grow longer, nor would we require that a given transaction should execute more quickly now than it did before. Hence transaction scale-up is the most relevant measure in this scenario.

17.11 Database systems are typically implemented as a set of processes (or threads) sharing a shared memory area.

a. How is access to the shared memory area controlled?

b. Is two-phase locking appropriate for serializing access to the data structures in shared memory? Explain your answer.

Answer:

a. A locking system is necessary to control access to the shared data structures. Since many database transactions only involve reading from a data structure, **reader-writer** locks should be used, allowing multiple processes to concurrently read from a data structure by using the lock in **shared** mode. A process that wishes to modify the data structure needs to obtain an **exclusive** lock on the data structure which prohibits concurrent access from other reading or writing processes.
Locks used for controlling access to shared memory data structures are typically not held in a two-phase manner (as described below), and are often called *latches* to distinguish them from locks held in a two-phase manner.

b. Shared memory areas are usually hot spots of contention, since every transaction needs to access the shared memory frequently. If such memory areas are locked in a two-phase manner, concurrency would be greatly reduced, reducing performance correspondingly. Instead, locks (latches) on shared memory data structures are released after performing operations on the data structure.
Serializability at a lower level, such as the exact layout of data on a page, or the structure of a B-tree or hash-index, is not important as long as the differences caused by non-serial execution are not visible at a higher level (typically, at the relational abstraction). Locks on tuples (or higher granularity) are retained in a two-phase manner to ensure serializability at the higher level.

17.12 Is it wise to allow a user process to access the shared memory area of a database system? Explain your answer.
Answer: No, the shared memory area may contain data that the user's process is not authorized to see, and thus allowing direct access to shared memory is a security risk.

17.13 What are the factors that can work against linear scaleup in a transaction processing system? Which of the factors are likely to be the most important in each of the following architectures: shared memory, shared disk, and shared nothing?
Answer: Increasing contention for shared resources prevents linear scale-up with increasing parallelism. In a shared memory system, contention for memory (which implies bus contention) will result in falling scale-up with

increasing parallelism. In a shared disk system, it is contention for disk and bus access which affects scale-up. In a shared-nothing system, interprocess communication overheads will be the main impeding factor. Since there is no shared memory, acquiring locks, and other activities requiring message passing between processes will take more time with increased parallelism.

17.14 Memory systems can be divided into multiple modules, each of which can be serving a separate request at a given time. What impact would such a memory architecture have on the number of processors that can be supported in a shared-memory system?

Answer: If all memory requests have to go through a single memory module, the memory module would become the bottleneck as the number of processors increases. After some point, adding processors will not result in any performnace improvement. However, if the memory system is itself able to run multiple requests in parallel, a larger number of processors can be supported in a shared-memory system, providing better speedup and/or scaleup.

17.15 Consider a bank that has a collection of sites, each running a database system. Suppose the only way the databases interact is by electronic transfer of money between themselves, using persistent messaging. Would such a system qualify as a distributed database? Why?

Answer: In a distributed database, it should be possible to run queries across sites, and to run transactions across sites using protocols such as two-phase commit. Each site provides an environment for execution of both global transactions initiated at remote sites and local transactions. The system described in the question does not have these properties, and hence it cannot qualify as a distributed database.