parallelism is achieved when there is no dependency between the operators that are executed in parallel. For instance, the two select operators of Fig. 1.7 can be executed in parallel. This form of parallelism is very attractive because there is no interference between the processors.

1.4.4 Scalability

In a distributed environment, it is much easier to accommodate increasing database sizes and bigger workloads. System expansion can usually be handled by adding processing and storage power to the network. Obviously, it may not be possible to obtain a linear increase in "power," since this also depends on the overhead of distribution. However, significant improvements are still possible. That is why distributed DBMSs have gained much interest in *scale-out* architectures in the context of cluster and cloud computing. Scale-out (also called *horizontal scaling*) refers to adding more servers, called "scale-out servers" in a loosely coupled fashion, to scale almost infinitely. By making it easy to add new component database servers, a distributed DBMS can provide scale-out.

1.5 Design Issues

In the previous section, we discussed the promises of distributed DBMS technology, highlighting the challenges that need to be overcome in order to realize them. In this section, we build on this discussion by presenting the design issues that arise in building a distributed DBMS. These issues will occupy much of the remainder of this book.

1.5.1 Distributed Database Design

The question that is being addressed is how the data is placed across the sites. The starting point is one global database and the end result is a distribution of the data across the sites. This is referred to as *top-down design*. There are two basic alternatives to placing data: *partitioned* (or *nonreplicated*) and *replicated*. In the partitioned scheme the database is divided into a number of disjoint partitions each of which is placed at a different site. Replicated designs can be either *fully replicated* (also called *fully duplicated*) where the entire database is stored at each site, or *partially replicated* (or *partially duplicated*) where each partition of the database is stored at more than one site, but not at all the sites. The two fundamental design issues are *fragmentation*, the separation of the database into partitions called *fragments*, and *distribution*, the optimum distribution of fragments.

A related problem is the design and management of system directory. In centralized DBMSs, the *catalog* contains metainformation (i.e., description) about the data.

In a distributed system, we have a directory that contains additional information such as where data is located. Problems related to directory management are similar in nature to the database placement problem discussed in the preceding section. A directory may be global to the entire distributed DBMS or local to each site; it can be centralized at one site or distributed over several sites; there can be a single copy or multiple copies. Distributed database design and directory management are topics of Chap. 2.

1.5.2 Distributed Data Control

An important requirement of a DBMS is to maintain data consistency by controlling how data is accessed. This is called *data control* and involves view management, access control, and integrity enforcement. Distribution imposes additional challenges since data that is required to check rules is distributed to different sites requiring distributed rule checking and enforcement. The topic is covered in Chap. 3.

1.5.3 Distributed Query Processing

Query processing deals with designing algorithms that analyze queries and convert them into a series of data manipulation operations. The problem is how to decide on a strategy for executing each query over the network in the most cost-effective way, however, cost is defined. The factors to be considered are the distribution of data, communication costs, and lack of sufficient locally available information. The objective is to optimize where the inherent parallelism is used to improve the performance of executing the transaction, subject to the above-mentioned constraints. The problem is NP-hard in nature, and the approaches are usually heuristic. Distributed query processing is discussed in detail in Chap. 4.

1.5.4 Distributed Concurrency Control

Concurrency control involves the synchronization of accesses to the distributed database, such that the integrity of the database is maintained. The concurrency control problem in a distributed context is somewhat different than in a centralized framework. One not only has to worry about the integrity of a single database, but also about the consistency of multiple copies of the database. The condition that requires all the values of multiple copies of every data item to converge to the same value is called *mutual consistency*.

The two general classes of solutions are *pessimistic*, synchronizing the execution of user requests before the execution starts, and *optimistic*, executing the requests and then checking if the execution has compromised the consistency of the database. Two fundamental primitives that can be used with both approaches are *locking*, which is based on the mutual exclusion of accesses to data items, and *timestamping*, where the transaction executions are ordered based on timestamps. There are variations of these schemes as well as hybrid algorithms that attempt to combine the two basic mechanisms.

In locking-based approaches deadlocks are possible since there is mutually exclusive access to data by different transactions. The well-known alternatives of prevention, avoidance, and detection/recovery also apply to distributed DBMSs. Distributed concurrency control is covered in Chap. 5.

1.5.5 Reliability of Distributed DBMS

We mentioned earlier that one of the potential advantages of distributed systems is improved reliability and availability. This, however, is not a feature that comes automatically. It is important that mechanisms be provided to ensure the consistency of the database as well as to detect failures and recover from them. The implication for distributed DBMSs is that when a failure occurs and various sites become either inoperable or inaccessible, the databases at the operational sites remain consistent and up-to-date. Furthermore, when the computer system or network recovers from the failure, the distributed DBMSs should be able to recover and bring the databases at the failed sites up-to-date. This may be especially difficult in the case of network partitioning, where the sites are divided into two or more groups with no communication among them. Distributed reliability protocols are the topic of Chap. 5.

1.5.6 Replication

If the distributed database is (partially or fully) replicated, it is necessary to implement protocols that ensure the consistency of the replicas, i.e., copies of the same data item have the same value. These protocols can be *eager* in that they force the updates to be applied to all the replicas before the transaction completes, or they may be *lazy* so that the transaction updates one copy (called the *master*) from which updates are propagated to the others after the transaction completes. We discuss replication protocols in Chap. 6.

1.5.7 Parallel DBMSs

As earlier noted, there is a strong relationship between distributed databases and parallel databases. Although the former assumes each site to be a single logical computer, most of these installations are, in fact, parallel clusters. This is the distinction that we highlighted earlier between single site distribution as in data center clusters and geo-distribution. Parallel DBMS objectives are somewhat different from distributed DBMSs in that the main objectives are high scalability and performance. While most of the book focuses on issues that arise in managing data in geo-distributed databases, interesting data management issues exist within a single site distribution as a parallel system. We discuss these issues in Chap. 8.

1.5.8 Database Integration

One of the important developments has been the move towards "looser" federation among data sources, which may also be heterogeneous. As we discuss in the next section, this has given rise to the development of multidatabase systems (also called *federated database systems*) that require reinvestigation of some of the fundamental database techniques. The input here is a set of already distributed databases and the objective is to provide easy access by (physically or logically) integrating them. This involves *bottom-up design*. These systems constitute an important part of today's distributed environment. We discuss multidatabase systems, or as more commonly termed now *database integration*, including design issues and query processing challenges in Chap. 7.

1.5.9 Alternative Distribution Approaches

The growth of the Internet as a fundamental networking platform has raised important questions about the assumptions underlying distributed database systems. Two issues are of particular concern to us. One is the re-emergence of peer-to-peer computing, and the other is the development and growth of the World Wide Web. Both of these aim at improving data sharing, but take different approaches and pose different data management challenges. We discuss peer-to-peer data management in Chap. 9 and web data management in Chap. 12.

1.5.10 Big Data Processing and NoSQL

The last decade has seen the explosion of "big data" processing. The exact definition of big data is elusive, but they are typically accepted to have four characteristics dubbed the "four V's": data is very high *volume*, is multimodal (*variety*), usually

comes at very high speed as data streams (*velocity*), and may have quality concerns due to uncertain sources and conflicts (*veracity*). There have been significant efforts to develop systems to deal with "big data," all spurred by the perceived unsuitability of relational DBMSs for a number of new applications. These efforts typically take two forms: one thread has developed general purpose computing platforms (almost always scale-out) for processing, and the other special DBMSs that do not have the full relational functionality, with more flexible data management capabilities (the so-called NoSQL systems). We discuss the big data platforms in Chap. 10 and NoSQL systems in Chap. 11.

1.6 Distributed DBMS Architectures

The architecture of a system defines its structure. This means that the components of the system are identified, the function of each component is specified, and the interrelationships and interactions among these components are defined. The specification of the architecture of a system requires identification of the various modules, with their interfaces and interrelationships, in terms of the data and control flow through the system.

In this section, we develop four "reference" architectures[2] for a distributed DBMS: client/server, peer-to-peer, multidatabase, and cloud. These are "idealized" views of a DBMS in that many of the commercially available systems may deviate from them; however, the architectures will serve as a reasonable framework within which the issues related to distributed DBMS can be discussed.

We start with a discussion of the design space to better position the architectures that will be presented.

1.6.1 Architectural Models for Distributed DBMSs

We use a classification (Fig. 1.8) that recognizes three dimensions according to which distributed DBMSs may be architected: (1) the autonomy of local systems, (2) their distribution, and (3) their heterogeneity. These dimensions are orthogonal as we discuss shortly and in each dimension we identify a number of alternatives. Consequently, there are 18 possible architectures in the design space; not all of these architectural alternatives are meaningful, and most are not relevant from the perspective of this book. The three on which we focus are identified in Fig. 1.8.

[2] A reference architecture is commonly created by standards developers to clearly define the interfaces that need to be standardized.