Chapter 1
Introduction

The current computing environment is largely distributed—computers are connected to Internet to form a worldwide distributed system. Organizations have geographically distributed and interconnected data centers, each with hundreds or thousands of computers connected with high-speed networks, forming mixture of distributed and parallel systems (Fig. 1.1). Within this environment, the amount of data that is captured has increased dramatically. Not all of this data is stored in database systems (in fact a small portion is) but there is a desire to provide some sort of data management capability on these widely distributed data. This is the scope of distributed and parallel database systems, which have moved from a small part of the worldwide computing environment a few decades ago to mainstream. In this chapter, we provide an overview of this technology, before we examine the details in subsequent chapters.

1.1 What Is a Distributed Database System?

We define a *distributed database* as *a collection of multiple, logically interrelated databases located at the nodes of a distributed system*. A *distributed database management system* (distributed DBMS) is then defined as *the software system that permits the management of the distributed database and makes the distribution transparent to the users*. Sometimes "distributed database system" (distributed DBMS) is used to refer jointly to the distributed database and the distributed DBMS. The two important characteristics are that data is logically interrelated and that it resides on a distributed system.

The existence of a distributed system is an important characteristic. In this context, we define a *distributed computing system* as a number of interconnected autonomous processing elements (PEs). The capabilities of these processing elements may differ, they may be heterogeneous, and the interconnections might be

M. T. Özsu, P. Valduriez, *Principles of Distributed Database Systems*,
https://doi.org/10.1007/978-3-030-26253-2_1

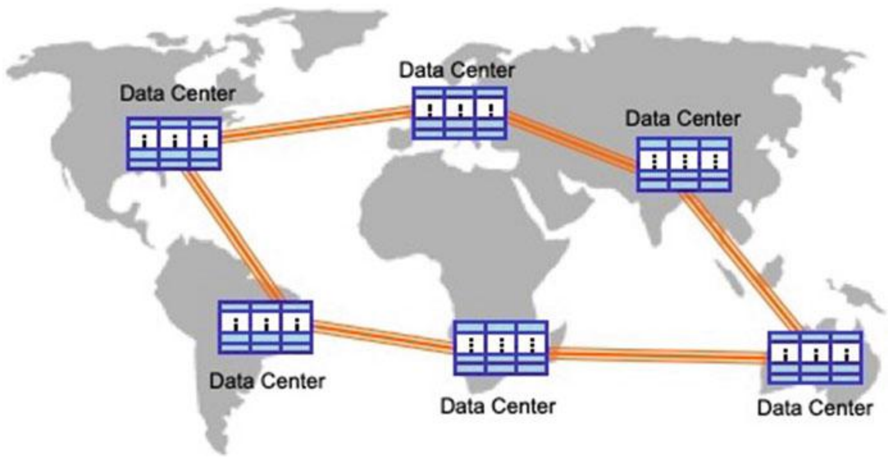

Fig. 1.1 Geographically distributed data centers

different, but the important aspect is that PEs do not have access to each other's state, which they can only learn by exchanging messages that incur a communication cost. Therefore, when data is distributed, its management and access in a logically integrated manner requires special care from the distributed DBMS software.

A distributed DBMS is not a "collection of files" that can be individually stored at each PE of a distributed system (usually called "site" of a distributed DBMS); data in a distributed DBMS is interrelated. We will not try to be very specific with what we mean by interrelated, because the requirements differ depending on the type of data. For example, in the case of relational data, different relations or their partitions might be stored at different sites (more on this in Chap. 2), requiring join or union operations to answer queries that are typically expressed in SQL. One can usually define a *schema* of this distributed data. At the other extreme, data in NoSQL systems (discussed further in Chap. 11) may have a much looser definition of interrelatedness; for example, it may be vertices of a graph that might be stored at different sites.

The upshot of this discussion is that a distributed DBMS is *logically integrated* but *physically distributed*. What this means is that a distributed DBMS gives the users the view of a unified database, while the underlying data is physically distributed.

As noted above, we typically consider two types of distributed DBMSs: geographically distributed (commonly referred to as *geo-distributed*) and single location (or single site) . In the former, the sites are interconnected by wide area networks that are characterized by long message latencies and higher error rates. The latter consist of systems where the PEs are located in close proximity allowing much faster exchanges leading to shorter (even negligible with new technologies) message latencies and very low error rates. Single location distributed DBMSs are typically characterized by computer clusters in one data center, and are commonly known as parallel DBMSs (and the PEs are referred to as “nodes” to distinguish from “sites”). As noted above, it is now quite common to find distributed DBMSs that have multiple single site clusters interconnected by wide area networks, leading to hybrid, multisite systems. For most of this book, we will focus on the problems of data management among the sites of a geo-distributed DBMS; we will focus on the problems of single site systems in Chaps. 8, 10, and 11 where we discuss parallel DBMSs, big data systems, and NoSQL/NewSQL systems.

1.2 History of Distributed DBMS

Before the advent of database systems in the 1960s, the prevalent mode of computation was one where each application defined and maintained its own data (Fig. 1.2). In this mode, each application defined the data that it used, its structure and access methods, and managed the file in the storage system. The end result was significant uncontrolled redundancy in the data, and high overhead for the programmers to manage this data within their applications.

Database systems allow data to be defined and administered centrally (Fig. 1.3). This new orientation results in *data independence*, whereby the application programs are immune to changes in the logical or physical organization of the data and vice versa. Consequently, programmers are freed from the task of managing and maintaining the data that they need, and the redundancy of the data can be eliminated (or reduced).

One of the original motivations behind the use of database systems was the desire to integrate the operational data of an enterprise and to provide integrated, thus controlled access to that data. We carefully use the term “integrated” rather

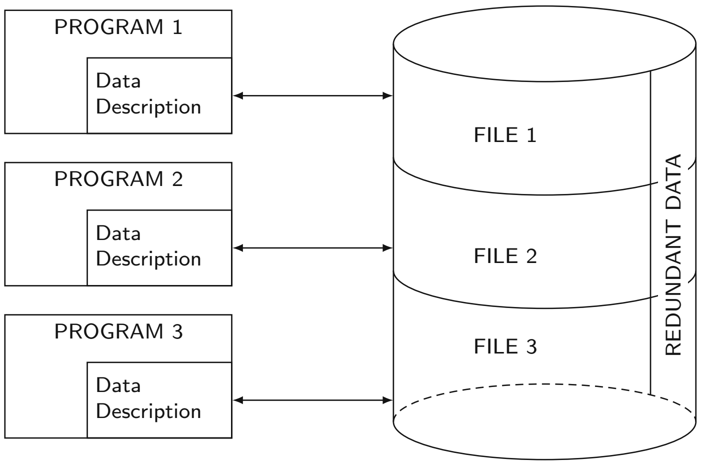

Fig. 1.2 Traditional file processing

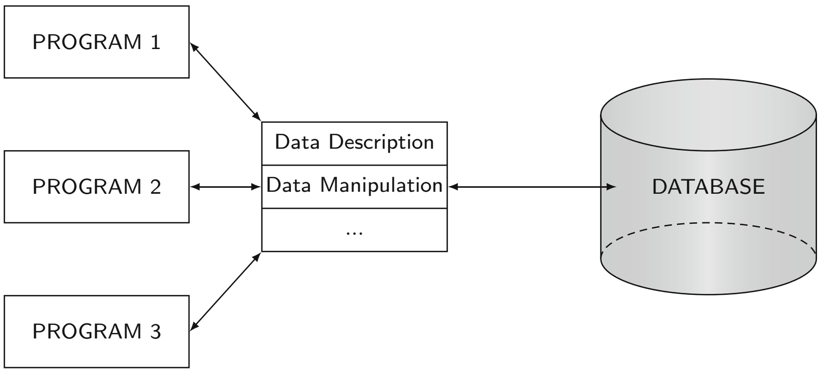

Fig. 1.3 Database processing

than "centralized" because, as discussed earlier, the data can physically be located on different machines that might be geographically distributed. This is what the distributed database technology provides. As noted earlier, this physical distribution can be concentrated at one geographic location or it can be at multiple locations. Therefore, each of the locations in Fig. 1.5 might be a data center that is connected to other data centers over a communication network. These are the types of distributed environments that are now common and that we study in this book.

Over the years, distributed database system architectures have undergone significant changes. The original distributed database systems such as Distributed INGRES and SDD-1 were designed as geographically distributed systems with very slow network connections; consequently they tried to optimize operations to reduce network communication. They were early *peer-to-peer systems* (P2P) in the sense that each site had similar functionality with respect to data management. With the development of personal computers and workstations, the prevailing distribution model shifted to *client/server* where data operations were moved to a back-end server, while user applications ran on the front-end workstations. These systems became dominant in particular for distribution at one particular location where the network speeds would be higher, enabling frequent communication between the clients and the server(s). There was a reemergence of P2P systems in the 2000s, where there is no distinction of client machines versus servers. These modern P2P systems have important differences from the earlier systems that we discuss later in this chapter. All of these architectures can still be found today and we discuss them in subsequent chapters.

The emergence of the World Wide Web (usually called the web) as a major collaboration and sharing platform had a profound impact on distributed data management research. Significantly more data was opened up for access, but this was not the well-structured, well-defined data DBMSs typically handle; instead, it is unstructured or semistructured (i.e., it has some structure but not at the level of a database schema), with uncertain provenance (so it might be "dirty" or unreliable), and conflicting. Furthermore, a lot of the data is stored in systems that are not easily accessible (what is called the *dark web*). Consequently, distributed data management efforts focus on accessing this data in meaningful ways.

This development added particular impetus to one thread of research that existed since the beginning of distributed database efforts, namely *database integration*. Originally, these efforts focused on finding ways to access data in separate databases (thus the terms *federated database* and *multidatabase*), but with the emergence of web data, these efforts shifted to virtual integration of different data types (and the term *data integration* became more popular). The term that is in vogue right now is *data lake* which implies that all of the data is captured in a logically single store, from which relevant data is extracted for each application. We discuss the former in Chap. 7 and the latter in Chaps. 10 and 12.

A significant development in the last ten years has been the emergence of cloud computing. Cloud computing refers to a computing model where a number of service providers make available shared and geo-distributed computing resources such that users can rent some of these resources based on their needs. Clients can rent the basic computing infrastructure on which they could develop their own software, but then decide on the operating system they wish to use and create virtual machines (VMs) to create the environment in which they wish to work—the so-called *Infrastructure-as-a-Service* (IaaS) approach. A more sophisticated cloud environment involves renting, in addition to basic infrastructure, the full computing platform leading to *Platform-as-a-Service* (PaaS) on which clients can develop their own software. The most sophisticated version is where service providers make available specific software that the clients can then rent; this is called *Software-as-a-Service* (SaaS). There has been a trend in providing distributed database management services on the cloud as part of SaaS offering, and this has been one of the more recent developments.

In addition to the specific chapters where we discuss these architectures in depth, we provide an overview of all of them in Sect. 1.6.1.2.

1.3 Data Delivery Alternatives

In distributed databases, data delivery occurs between sites—either from server sites to client sites in answer to queries or between multiple servers. We characterize the data delivery alternatives along three orthogonal dimensions: *delivery modes*, *frequency*, and *communication methods*. The combinations of alternatives along each of these dimensions provide a rich design space.

The alternative delivery modes are pull-only, push-only, and hybrid. In the *pull-only* mode of data delivery, the transfer of data is initiated by a pull (i.e., request) from one site to a data provider—this may be a client requesting data from a server or a server requesting data from another server. In the following we use the terms "receiver" and "provider" to refer to the machine that received the data and the machine that sends the data, respectively. When the request is received by the provider, the data is located and transferred. The main characteristic of pull-based delivery is that receivers become aware of new data items or updates at the provider only when they explicitly poll. Also, in pull-based mode, providers must be interrupted continuously to deal with requests. Furthermore, the data that receivers can obtain from a provider is limited to when and what clients know to ask for. Conventional DBMSs offer primarily pull-based data delivery.

In the *push-only* mode of data delivery, the transfer of data from providers is initiated by a push without a specific request. The main difficulty of the push-based approach is in deciding which data would be of common interest, and when to send it to potentially interested receivers—alternatives are periodic, irregular, or conditional. Thus, the usefulness of push depends heavily upon the accuracy of a provider to predict the needs of receivers. In push-based mode, providers disseminate information to either an unbounded set of receivers (random broadcast) who can listen to a medium or selective set of receivers (multicast), who belong to some categories of recipients.

The *hybrid* mode of data delivery combines the pull and push mechanisms. The persistent query approach (see Sect. 10.3) presents one possible way of combining the pull and push modes, namely: the transfer of data from providers to receivers is first initiated by a pull (by posing the query), and the subsequent transfer of updated data is initiated by a push by the provider.

There are three typical frequency measurements that can be used to classify the regularity of data delivery. They are periodic, conditional, and ad hoc (or irregular).

In *periodic delivery*, data is sent from the providers at regular intervals. The intervals can be defined by system default or by receivers in their profiles. Both pull and push can be performed in periodic fashion. Periodic delivery is carried out on a regular and prespecified repeating schedule. A request for a company's stock price every week is an example of a periodic pull. An example of periodic push is when an application can send out stock price listing on a regular basis, say every morning. Periodic push is particularly useful for situations in which receivers might not be available at all times, or might be unable to react to what has been sent, such as in the mobile setting where clients can become disconnected.

In *conditional delivery*, data is sent by providers whenever certain conditions specified by receivers in their profiles are satisfied. Such conditions can be as simple as a given time span or as complicated as event-condition-action rules. Conditional delivery is mostly used in the hybrid or push-only delivery systems. Using conditional push, data is sent out according to a prespecified condition, rather than any particular repeating schedule. An application that sends out stock prices only when they change is an example of conditional push. An application that sends out a balance statement only when the total balance is 5% below the predefined balance threshold is an example of hybrid conditional push. Conditional push assumes that changes are critical to the receivers who are always listening and need to respond to what is being sent. Hybrid conditional push further assumes that missing some update information is not crucial to the receivers.

Ad hoc delivery is irregular and is performed mostly in a pure pull-based system. Data is pulled from providers in an ad hoc fashion in response to requests. In contrast, periodic pull arises when a requestor uses polling to obtain data from providers based on a regular period (schedule).

The third component of the design space of information delivery alternatives is the communication method. These methods determine the various ways in which providers and receivers communicate for delivering information to clients. The alternatives are unicast and one-to-many. In *unicast*, the communication from a provider to a receiver is one-to-one: the provider sends data to one receiver using a particular delivery mode with some frequency. In *one-to-many*, as the name implies, the provider sends data to a number of receivers. Note that we are not referring here to a specific protocol; one-to-many communication may use a multicast or broadcast protocol.

We should note that this characterization is subject to considerable debate. It is not clear that every point in the design space is meaningful. Furthermore, specification of alternatives such as conditional **and** periodic (which may make sense) is difficult. However, it serves as a first-order characterization of the complexity of emerging distributed data management systems. For the most part, in this book, we are concerned with pull-only, ad hoc data delivery systems, and discuss push-based and hybrid modes under streaming systems in Sect. 10.3.

1.4 Promises of Distributed DBMSs

Many advantages of distributed DBMSs can be cited; these can be distilled to four fundamentals that may also be viewed as promises of distributed DBMS technology: transparent management of distributed and replicated data, reliable access to data through distributed transactions, improved performance, and easier system expansion. In this section, we discuss these promises and, in the process, introduce many of the concepts that we will study in subsequent chapters.

1.4.1 Transparent Management of Distributed and Replicated Data

Transparency refers to separation of the higher-level semantics of a system from lower-level implementation issues. In other words, a transparent system "hides" the implementation details from users. The advantage of a fully transparent DBMS is the high level of support that it provides for the development of complex applications. Transparency in distributed DBMS can be viewed as an extension of the data independence concept in centralized DBMS (more on this below).

Let us start our discussion with an example. Consider an engineering firm that has offices in Boston, Waterloo, Paris, and San Francisco. They run projects at each of these sites and would like to maintain a database of their employees, the projects,