A storage model describes the layout of a data structure in physical storage; a data model captures the

most important logical aspects of a data structure in a database. The physical storage can be a local disk,

a removable media, or storage accessible via a network.

Two abstract models of storage are commonly used: cell storage and journal storage. Cell storage

assumes that the storage consists of cells of the same size and that each object fits exactly in one cell. This

model reflects the physical organization of several storage media; the primary memory of a computer

is organized as an array of memory cells, and a secondary storage device (e.g., a disk) is organized in

sectors or blocks read and written as a unit. read/write coherence and before-or-after atomicity

are two highly desirable properties of any storage model and in particular of cell storage (see Figure 8.1).

Journal storage is a fairly elaborate organization for storing composite objects such as records

consisting of multiple fields. Journal storage consists of a manager and cell storage, where the entire history of a variable is maintained, rather than just the current value. The user does not have direct

access to the cell storage; instead the user can request the journal manager to (i) start a new action;

(ii) read the value of a cell; (iii) write the value of a cell; (iv) commit an action; or (v) abort an

action. The journal manager translates user requests to commands sent to the cell storage: (i) read a

cell; (ii) write a cell; (iii) allocate a cell; or (iv) deallocate a cell.

In the context of storage systems, a log contains a history of all variables in cell storage. The information

about the updates of each data item forms a record appended at the end of the log. A log provides

authoritative information about the outcome of an action involving cell storage; the cell storage can be

reconstructed using the log, which can be easily accessed – we only need a pointer to the last record.

An all-or-nothing action first records the action in a log in journal storage and then installs the

change in the cell storage by overwriting the previous version of a data item (see Figure 8.2). The log is

always kept on nonvolatile storage (e.g., disk) and the considerably larger cell storage resides typically

on nonvolatile memory, but can be held in memory for real-time access or using a write-through cache.

Many cloud applications must support online transaction processing and have to guarantee the correctness

of the transactions. Transactions consist of multiple actions; for example, the transfer of funds

from one account to another requires withdrawing funds from one account and crediting it to another.

The system may fail during or after each one of the actions, and steps to ensure correctness must be

taken. Correctness of a transaction means that the result should be guaranteed to be the same as though

the actions were applied one after another, regardless of the order. More stringent conditions must sometimes be observed; for example, banking transactions must be processed in the order in which they

are issued, the so-called external time consistency. To guarantee correctness, a transaction-processing

system supports all-or-nothing atomicity, discussed in Section 2.10.

A file system consists of a collection of directories. Each directory provides information about a

set of files. Today high-performance systems can choose among three classes of file system: network

file systems (NFSs), storage area networks (SANs), and parallel file systems (PFSs). The NFS is very

popular and has been used for some time, but it does not scale well and has reliability problems; an

NFS server could be a single point of failure.

Advances in networking technology allow the separation of storage systems from computational

servers; the two can be connected by a SAN. SANs offer additional flexibility and allow cloud servers

to deal with nondisruptive changes in the storage configuration. Moreover, the storage in a SAN can

be pooled and then allocated based on the needs of the servers; pooling requires additional software

and hardware support and represents another advantage of a centralized storage system. A SAN-based

implementation of a file system can be expensive, since each node must have a Fibre Channel adapter

to connect to the network.

Parallel file systems are scalable, are capable of distributing files across a large number of nodes, and

provide a global naming space. In a parallel data system, several I/O nodes serve data to all computational

nodes and include a metadata server that contains information about the data stored in the I/O nodes.

The interconnection network of a parallel file system could be a SAN.

Most cloud applications do not interact directly with file systems but rather through an application

layer that manages a database. A database is a collection of logically related records. The software

that controls the access to the database is called a database management system (DBMS). The main

functions of a DBMS are to enforce data integrity, manage data access and concurrency control, and

support recovery after a failure.

A DBMS supports a query language, a dedicated programming language used to develop database

applications. Several database models, including the navigational model of the 1960s, the relational

model of the 1970s, the object-oriented model of the 1980s, and the NoSQL model of the first decade

of the 2000s, reflect the limitations of the hardware available at the time and the requirements of the

most popular applications of each period.

Most cloud applications are data intensive and test the limitations of the existing infrastructure. For

example, they demand DBMSs capable of supporting rapid application development and short time to

market. At the same time, cloud applications require low latency, scalability, and high availability and

demand a consistent view of the data.

These requirements cannot be satisfied simultaneously by existing database models; for example,

relational databases are easy to use for application development but do not scale well. As its name

implies, the NoSQL model does not support SQL as a query language and may not guarantee the

atomicity, consistency, isolation, durability (ACID) properties of traditional databases. NoSQL usually

guarantees the eventual consistency for transactions limited to a single data item. The NoSQL model is

useful when the structure of the data does not require a relational model and the amount of data is very

large. Several types of NoSQL database have emerged in the last few years. Based on the way the NoSQL

databases store data, we recognize several types, such as key-value stores, BigTable implementations,

document store databases, and graph databases.

Replication, used to ensure fault tolerance of large-scale systems built with commodity components,

requires mechanisms to guarantee that all replicas are consistent with one another. This is another

example of increased complexity of modern computing and communication systems due to physical

characteristics of components, a topic discussed in Chapter 10. Section 8.7 contains an in-depth analysis

of a service implementing a consensus algorithm to guarantee that replicated objects are consistent.