ARTICLE 7

Distributed Databases, Blockchain, Parallel Databases, and Cloud Computing

Abhishek Sharma
Loyola University Chicago

Svetlozar Nestorov
Loyola University Chicago

Nenad Jukić
Loyola University Chicago

In this article, we will give a brief overview of distributed databases, blockchain, parallel databases, and cloud computing.

A **distributed database system (DDBS)** is a database system in which the database is distributed among separate computers. A DDBS allows for the sharing of data and offers local database control and autonomy. A distributed database mirrors the organizational structures that are naturally distributed. As an example of a distributed database, consider an enterprise that consists of a headquarters and multiple branches. The enterprise needs to maintain information about the customers of each branch, as well as administrative information about its headquarters. With the distributed database shown in Figure 1, each branch can maintain information in its local database about its customers and may also access customer data from other branches without having to store that information in its local database. In addition, the headquarters of the enterprise can maintain administrative information about all the branches in its enterprise. Branches may access some of this administrative data from the database at the headquarters location, while other administrative data may be replicated in the database at each branch location.

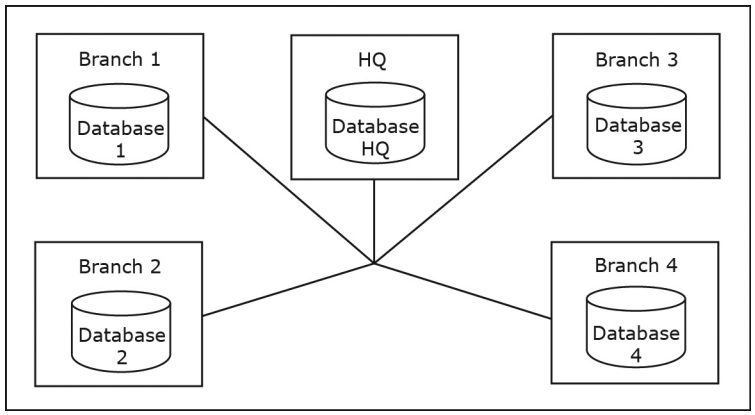

Figure 1 An example of a distributed database system.

End users should not be concerned with the distributed nature of the system. In other words,

the users should not have to know if the data being accessed is stored locally or remotely at an entirely different geographical location. The property of a DDBS that ensures that the end users can use the distributed database in the same fashion as if it were not distributed (i.e., without knowing the details of the distribution of the data or even if the data is distributed) is called **distribution transparency**.

In a DDBS, the data from the database is distributed among a number of separate computers that do not share any processing or storage devices, and each computer contains its own DBMS. The computers in a DDBS are connected by a network, so they may reside in the same building or be dispersed across different continents. In a **homogeneous DDBS**, all the computers run the same DBMS, while in a **heterogeneous DDBS**, different DBMS can run on different computers (e.g., due to a recent merger between two companies). In a heterogeneous DDBS, for example, one machine in the DDBS can use Oracle and another can use PostgreSQL as its DBMS.

A distributed database makes certain performance advantages possible. A distributed database allows users of the database to access different parts of the database at the same time without interfering with one another. This is particularly beneficial if most of the queries within the entire DDBS involve local users using local data, such as when most of the queries on the data in Branch 1 involve local users at the Branch 1 site, and most of the queries on the data in Branch 2 involve local users at the Branch 2 site, and so on. When compared to all queries being processed on one database (as is the case with the nondistributed database), a distributed database composed of databases stored at multiple locations results in fewer queries per computer. This allows for faster processing of the data in cases where most queries are local. In those cases, the data accessed is most often stored locally for each computer, which decreases access time. In addition, a distributed database can provide increased reliability and availability if the data is replicated at more than one location. If there is a problem accessing data from the database at one location, the data can be accessed from another location where the data is replicated.

It is important to note that distributed databases, when compared to nondistributed databases, are more complex. Additional functions are needed to perform tasks, such as keeping track of the location of the data and any replicas, determining which copy of the data to use for a query, and making sure updates are applied to all copies of the data. A distributed directory containing information about all local as well as all remote data in the database has to be maintained. One example of the increased complexity occurs when processing a query across the entire distributed database. To process such a query, it is necessary to determine the location of the data needed for the query in order to identify the parts of the query that require local data and the parts of the query that require data from remote locations. When data needed for the query is at a remote location, the part of the query that requires remote data is sent to the appropriate remote location to be run. Once the results of the query from the remote locations are returned, they are combined with the local results.

Database fragmentation is a strategy for distributing the data across different locations in a distributed database. In a distributed database, entire tables can be stored at different locations or fragments of a table can be stored at different locations. The database can be fragmented horizontally or vertically.

In **horizontal fragmentation**, subsets of the records from a table are stored at different locations in the DDBS. All the columns in the table are stored at a location, but different subsets of records from the table are stored at different locations. To reconstruct the complete table, all the fragments must be brought together using the union operator. Figure 2 shows an example of a horizontally fragmented table.

EMPLOYEE (Not fragmented)

EmpID	EmpName	EmpGender	EmpPhone	EmpBdate
0001	Joe	M	x-234	1/11/1985
0002	Sue	F	x-345	2/7/1983
0003	Amy	F	x-456	8/4/1990
0004	Pat	F	x-567	3/8/1971
0005	Mike	M	x-678	5/5/1965

EMPLOYEE (Horizontally fragmented - Location A)

EmpID	EmpName	EmpGender	EmpPhone	EmpBdate
0001	Joe	M	x-234	1/11/1985
0002	Sue	F	x-345	2/7/1983
0003	Amy	F	x-456	8/4/1990

EMPLOYEE (Horizontally fragmented - Location B)

EmpID	EmpName	EmpGender	EmpPhone	EmpBdate
0001	Joe	M	x-234	1/11/1985
0004	Pat	F	x-567	3/8/1971
0005	Mike	M	x-678	5/5/1965

Figure 2 An example of horizontal fragmentation.

In this example, the following query would create a union of two fragments (each containing three records) and thereby reconstruct the complete table (containing five records, as the fragments share the same record for employee Joe):

```
SELECT * FROM employee_location_a
UNION
SELECT * FROM employee_location_b;
```

The distribution transparency property of the DDBS would allow the user to simply issue this query as

```
SELECT * FROM employee
```

In **vertical fragmentation**, subsets of the columns of a table are stored at different locations in the DDBS. All the records from the table are stored at every location, but only some of the columns. Since only a subset of columns is stored at a location, it is important to include the primary key from the table as one of the columns in each vertical fragment. To reconstruct the complete table, all the fragments have to be joined together, using the primary key as the JOIN condition.

Figure 3 shows an example of a vertically fragmented table. In this example, the following query would create a join of two fragments and thereby reconstruct the complete table:

```
SELECT      a.empid, a.empname, b.empgender, a.empphone, b.empbdate
FROM        employee_location_a a, employee_location_b b
WHERE       a.empid = b.empid;
```

The distribution transparency property of the DDBS would allow the user to simply issue this query as

```
SELECT * FROM employee
```

EMPLOYEE (Not fragmented)

EmpID	EmpName	EmpGender	EmpPhone	EmpBdate
0001	Joe	M	x-234	1/11/1985
0002	Sue	F	x-345	2/7/1983
0003	Amy	F	x-456	8/4/1990
0004	Pat	F	x-567	3/8/1971
0005	Mike	M	x-678	5/5/1965

EMPLOYEE (Vertically fragmented - Location A)

EmpID	EmpName	EmpPhone
0001	Joe	x-234
0002	Sue	x-345
0003	Amy	x-456
0004	Pat	x-567
0005	Mike	x-678

EMPLOYEE Vertically fragmented - Location B)

EmpID	EmpGender	EmpBdate
0001	M	1/11/1985
0002	F	2/7/1983
0003	F	8/4/1990
0004	F	3/8/1971
0005	M	5/5/1965

Figure 3 An example of vertical fragmentation.

It is possible to combine both horizontal and vertical fragmentation strategies within the same table. Combining horizontal and vertical fragmentation is referred to as **mixed fragmentation**. Figure 4 shows an example of a table fragmented using mixed fragmentation.

In this example, the following query would reconstruct the complete table:

```
SELECT      a.empid, a.empname, c.empgender, a.empphone, c.empbdate
FROM        employee_location_a a, employee_location_c c
WHERE       a.empid = c.empid
UNION
SELECT      b.empid, b.empname, c.empgender, b.empphone, c.empbdate
FROM        employee_location_b b, employee_location_c c
WHERE       b.empid = c.empid;
```

The distribution transparency property of the DDBS would allow the user to simply issue this query as

```
SELECT * FROM employee
```

EMPLOYEE (Not fragmented)

EmpID	EmpName	EmpGender	EmpPhone	EmpBdate
0001	Joe	M	x-234	1/11/1985
0002	Sue	F	x-345	2/7/1983
0003	Amy	F	x-456	8/4/1990
0004	Pat	F	x-567	3/8/1971
0005	Mike	M	x-678	5/5/1965

EMPLOYEE (Location A)

EmpID	EmpName	EmpPhone
0001	Joe	x-234
0002	Sue	x-345
0003	Amy	x-456

EMPLOYEE (Location B)

EmpID	EmpName	EmpPhone
0001	Joe	x-234
0004	Pat	x-567
0005	Mike	x-678

EMPLOYEE (Location C)

EmpID	EmpGender	EmpBdate
0001	M	1/11/1985
0002	F	2/7/1983
0003	F	8/4/1990
0004	F	3/8/1971
0005	M	5/5/1965

Figure 4 An example of mixed fragmentation.

Data replication occurs when more than one copy of the data is stored at different locations in a distributed database. When the entire database is replicated at each location in the distributed system, it is called a **fully replicated distributed database**. A type of replication in which some of the data is replicated at multiple locations while other parts are not is called **partial replication**.

The main advantage of data replication is the quicker performance of queries (more data can be processed locally) and the availability of data even when a computer that is part of a DDBS is not functioning (as long as another computer contains the same data that is stored on the nonfunctioning computer). While data replication can increase query performance and availability of the data, it can also complicate the update process. When one copy of the data is updated, a strategy must be in place to make sure all the other copies are updated as well. In one update strategy, one of the copies of the data is designated as a "distinguished" copy, whose location is responsible for coordinating updates to all other copies. Updated data cannot be accessed until the coordinator ensures that all copies have been successfully updated. Another update strategy, called "quorum" or "voting," does not have a coordinator but instead requires a majority of the copies to be updated before that data can be accessed.

A **federated database** is a type of distributed database system that comprises a collection of preexisting databases that are connected into one system. The preexisting databases remain autonomous, but they cooperate and share data in order to form a new database. One example is a federated library database that comprises the library databases at different universities across the country. While each library has its own database containing information about books stored in that library, the libraries agree to form a federated database. Users can then access information from all the university libraries that are in the federated database. Federated databases do not necessarily provide transparency. A federated database requires a global view of the federation that can be used when accessing the entire system, since each database in the federation has its own schema.

Blockchain

Blockchain is a mechanism for keeping track of data in a completely decentralized, distributed fashion, without a centralized coordinating system. Blockchain contains a continuously growing list of records where data is stored on participating computers (nodes). This continuously growing list of records is distributed on the participating nodes and is heavily replicated to avoid any possibility of loss of data. All nodes adhere to an agreed protocol of communication that is based on a resource-intensive and time-consuming cryptography (there is no trust between the participants). Blockchain is designed to allow data only to be added and viewed. It is not designed for alterations or deletions of previously entered data, since blockchain is designed to be immutable.

Blockchain came into prominence in the context of financial transactions using virtual, nonpaper currencies (cryptocurrencies) such as Bitcoin. The main purpose of these blockchain implementations was to alleviate the problem of establishing truthfulness of a transaction and eliminate the possibility of a party deceitfully using the same virtual currency token to receive multiple goods, because virtual currencies cannot be tracked like regular currencies. This problem is called "double spending" or spending the same money multiple times for malicious purposes.

Transactions can be conducted only if there is a way to authenticate the veracity of transactions. A seller cannot sell unless he or she has bought and/or possesses the goods, a payer cannot pay unless he or she has money, and the matching amount of money is irreversibly exhausted when the payer receives the matching goods. Traditionally, this responsibility is borne by a central authority like a central bank. (In the case of cash transactions, a central bank guarantees the paper currency as a valid form of payment.)

When it comes to electronic transactions, a central authority like a bank maintains a ledger that has detailed transactions that match one another, and hence subsequent dependent transactions can only happen if they align with the corresponding previous transactions (also called debit and credit). If a central authority is not present and there are no ledgers to trace back the transactions, there is no mechanism for establishing the veracity of a transaction or the provider of a good in order to determine if that payment is real or fake.

Blockchain solves this problem in the virtual world where there is no centralized coordinating system. Instead of a central authority maintaining a ledger, an online ledger is maintained by the community of participants. However, since this ledger is public, without appropriate measures, it can be manipulated to create fake transactions such as a receipt of goods without actual payment. There are two key problems to solve: (1) authenticating the transaction, and (2) preventing changes to previous transactions.

Blockchain uses cryptography to solve the problem of authenticating the transactions, which in itself is not a new thing. For example, various existing exchanges, including emails, use cryptography to ensure an email is coming from an authentic party. Using cryptography, every originating message is signed/encrypted by the originating user's private key (that no one else knows). The recipient has the public key of the originating user and it is used to authenticate/decrypt that the message has definitely come only from the originating user. Similarly, the message is sent back by the recipient (encrypted by the originating user's public key) and can be decrypted by the originating user's private key. This mechanism is also called *public key infrastructure*, or *PKI* for short.

In order to prevent changes to the previous transactions, blockchain goes a step further and chunks the transactions into blocks of message, which are then converted in cryptographic signatures called hashes. These hashes are unique for a given set of input data (content of the block), and if any change is made to any transactions in the block (including rearranging their order), it will lead to a different random hash. Henceforth, if a hash of a block is known to participants, no one can change the transactions inside the block without changing the hash. Therefore, any malicious changes will be caught by the participants.

Since blockchain does not have centralized authority to authenticate and certify the transactions, this responsibility is left to a decentralized group of volunteer miners. Their primary job is to verify the authenticity/validity of the transaction in a given block of transactions. These miners are rewarded for their work. However, since there could be multiple miners who could do this relatively trivial job simultaneously (and there may be some who would do it for nefarious purposes), this task also involves calculating a computationally expensive problem called "proof of work."

Along with transactions and previous block's hash value, an integer value called *Nounce* is also included in the block of transaction. Nounce is a number that is deduced by a miner (or a similar process in other industry vertical/use cases) that must lead to a specific attribute about a hash (such as, for example, the first three digits must be 111). This random search for appropriate Nounce is also called "proof of work."

The resulting hash will become the hash of the next block, integrating the blockchain. The miner who not only validates the transaction but also solves this problem is rewarded for the work, and the block is added to the blockchain. This computationally expensive proof of work ensures that any changes in historical data will be computationally impossible. This design further prevents any bad actors, such as hackers, from injecting fake transactions and redoing the hashes and blocks. Users can review the data but cannot make any changes to the historical data due to unsurmountable computational costs.

To summarize, blockchain is a distributed public ledger, maintained across many computers participating in a peer-to-peer network, adhering to an agreed-upon protocol of communication, and working on a resource-intensive and time-consuming cryptography to mitigate the risk of an environment where there is no central authority and there is no trust between the participants.

These characteristics make blockchain an ideal solution for the implementation of virtual currency, and niche applications in other industries and businesses, such as specialized data management for certain types of supply chain and accounting processes (e.g., keeping track of conflict-free diamonds, unconventional loans, and cross-border payments, etc.). However, it is important to note that much of traditional business and organizational data exists in environments with a central authority and/or some implicit trust between the users of the information. In such environments, the fully implemented blockchain as described here is not a mainstream alternative for storing and processing data, because the intense cost of processing every transaction is not warranted.

Parallel Databases

The computers in a **parallel database** system work on the same task and perform the same operations simultaneously on different portions or data slices of the same data set. Once individual computation is completed on these computers (also known as data nodes), the corresponding output is sent to a master computer (master node), which then merges and produces the result to the end user. In other words, the computers work in parallel to solve a given query. Modern parallel databases utilize **massively parallel processing (MPP)** technology. "MPP" is an industry term referring to a large number of separate computer processors running in parallel to execute a single program. There are various architectural approaches to parallel computing that differ in whether computer processors share or maintain their own disk storage and memory. Depending on the approach, the computer processors in a parallel system may share disk storage and memory or have their own memory and share disk storage. A parallel system in which each processor has its own memory and disk storage is referred to as a **shared-nothing MPP architecture**.

The parallelization of database operations can improve performance greatly when dealing with large amounts of data. This is why many data warehouse systems are implemented using parallel RDBMS systems, such as Teradata or Greenplum.

Chapter 10 of *Database Systems: Introduction to Databases and Data Warehouses (Edition 3.0)* illustrates the use of parallel computation for the implementation of the MapReduce method for dealing with unstructured data. Parallel computations are also very common in **cloud computing systems**, which use the internet to deliver data storage and processing services hosted by a service provider.

Cloud Computing

Cloud systems have emerged as a computing model as well as a business model, providing computing resources to users upon request and allowing users to increase or decrease the amount of resources requested based on their needs. The computing resources provided can range from computer software to computer hardware and can even be a particular computing platform. By utilizing a cloud, an organization does not have to invest in computer hardware and support an IT staff.

Some of the characteristics of a cloud are its on-demand access to computing resources, pay-per-use model, elasticity, virtualization, and distributed/highly parallel approach.

Clouds provide resources on-demand to customers as they are requested. Customers pay per use because they are only charged for the resources they use. Since customers can request resources as they are needed, they do not have the cost of maintaining unneeded resources. For example, promotions, sales, festivals, and special occasions in a consumer market can create a large spike in the workload of data applications. Deployment of available resources to handle that spike could be the wrong strategic decision for any company because those resources are unused during the rest of the year.

Cloud service providers tend to offer services that can be grouped into three categories: infrastructure as a service, platform as a service, and software as a service.

Infrastructure as a service (IaaS) provides hardware as standardized services over the network. Customers can rent resources, such as server space, network equipment, memory, processing cycles, and storage space.

Platform as a service (PaaS) supplies all the resources required by a customer to build applications. It encapsulates a layer of software to create a platform by integrating an operating system, middleware, application software, and a development environment. The customer can interact with the platform through an interface, and the platform performs actions to maintain and scale itself according to the customer's needs.

Software as a service (SaaS) features an application offered as a service on-demand, which is hosted and operated over the internet for use by customers. A single instance of the software running on the cloud can be used simultaneously by multiple clients (multitenant).

Database as a service (DaaS) is another specialized version of SaaS that is specifically developed for database management in clouds.

As the amount of data being stored is increasing dramatically, cloud architectures can solve some of the key difficulties faced in large-scale data processing. A cloud computing environment can allow databases to auto-scale up and down based on dynamic workloads. Clouds are viewed as well suited for analytical data management applications, which handle historical data from multiple operational databases and require few or no updates. Operational databases present more challenges when deployed in a cloud. Unlike analytical databases, operational databases require updates to the data. Maintaining consistent values when data is updated can become complex and can affect performance when data is replicated over a wide geographic area.

Clouds are described as "elastic" because they can accommodate an increase in demand for resources as well as a decrease in demand. Clouds use virtualization to allow multiple tenants to use the same equipment at the same time. A single server in a cloud can provide multiple operating instances so that more than one client can run an application using a different operating system on the same machine. The virtualization software can emulate

many physical machines and platforms for the customer. For example, a customer can choose a particular operating system on which to run his application.

Clouds are typically distributed and can comprise multiple data centers distributed across large geographical areas. Solving problems in parallel exploits the advantages of a cloud, as multiple machines can be used to solve a problem quickly in parallel. Instead of expensive servers, many clouds are composed of commodity or off-the-shelf computers. The possibility of failure of any component of a cloud is built into the computing model of a cloud. Processes can be replicated so that any processing failure can result in a quick recovery. Similarly, data reliability and availability are usually achieved by creating a certain number of replicas of the data distributed over different geographical areas.

There are certain security and privacy issues associated with storing data in a cloud. Moving data off the premises and storing it in a third-party vendor's servers potentially increases security risks. Storing an enterprise's data in a cloud requires trust in the host. In a cloud, data can be stored in any location around the globe. Different countries have different rules and regulations regarding the rightful access to data. By law, any government can require the right to access data stored in that country, in which case data may be handed over without any notification to the owner of the data. Accountability issues related to incorrect billing or nonpayment of bills may also arise. In spite of these challenges, cloud computing is a growing trend and is being adopted by an increasing number of companies and organizations worldwide.