CHAPTER 3

BIG DATA STORAGE AND ANALYTICS

3.1 BIG DATA STORAGE CONCEPTS [26]

Data acquired from external sources is often not in a format or structure that can be directly processed. To overcome these incompatibilities and prepare data for storage and processing, data wrangling is necessary. Data wrangling includes steps to filter, cleanse and otherwise prepare the data for downstream analysis. From a storage perspective, a copy of the data is first stored in its acquired format, and, after wrangling, the prepared data needs to be stored again. Typically, storage is required whenever the following occurs:

• external datasets are acquired, or internal data will be used in a Big Data environment
• data is manipulated to be made amenable for data analysis
• data is processed via an ETL activity, or output is generated as a result of an analytical operation

Due to the need to store Big Data datasets, often in multiple copies, innovative storage strategies and technologies have been created to achieve cost-effective and highly scalable storage solutions. In order to understand the underlying mechanisms behind Big Data storage technology, the following topics are introduced in this chapter:
• clusters
• file systems and distributed files systems
• NoSQL
• sharding
• replication
• CAP theorem
• ACID
• BASE

3.1.1 Clusters [26]

In computing, a cluster is a tightly coupled collection of servers, or nodes. These servers usually have the same hardware specifications and are connected together via a network to work as a single unit, as shown in Figure 3.1. Each node in the cluster has its own dedicated resources, such as memory, a processor, and a hard drive. A cluster can execute a task by splitting it into small pieces and distributing their execution onto different computers that belong to the cluster.

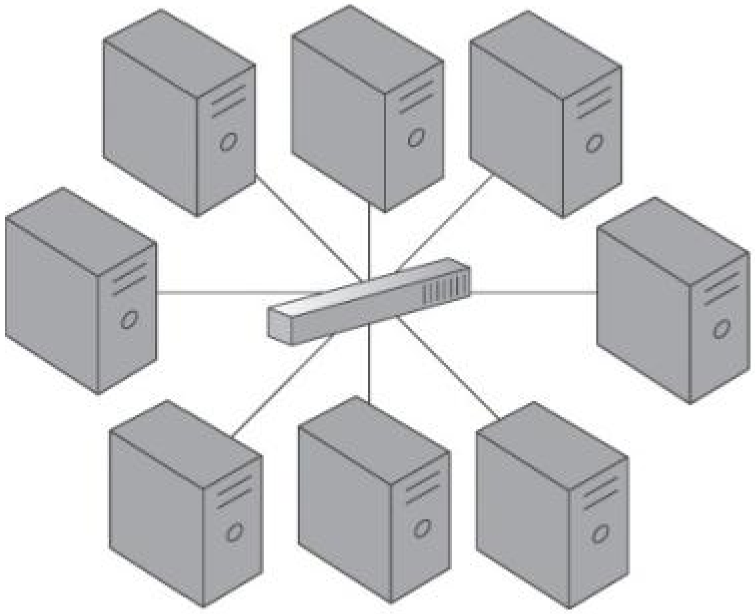


Figure 3.1 The symbol used to represent a cluster [26]

3.1.2 File Systems and Distributed File Systems [26]

A file system is the method of storing and organizing data on a storage device, such as flash drives, DVDs and hard drives. A file is an atomic unit of storage used by the file system to store data. A file system provides a logical view of the data stored on the storage device and presents it as a tree structure of directories and files as pictured in Figure 3.2. Operating systems employ file systems to store and retrieve data on behalf of applications. Each operating system provides support for one or more file systems, for example NTFS on Microsoft Windows and ext on Linux.

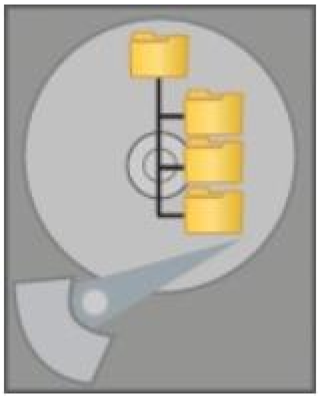


Figure 3.2 The symbol used to represent a file system [26]

A distributed file system is a file system that can store large files spread across the nodes of a cluster, as illustrated in Figure 3.3. To the client, files appear to be local; however, this is only a logical view as physically the files are distributed throughout the cluster. This local view is presented via the distributed file system and it enables the files to be accessed from multiple locations. Examples include the Google File System (GFS) and Hadoop Distributed File System (HDFS).

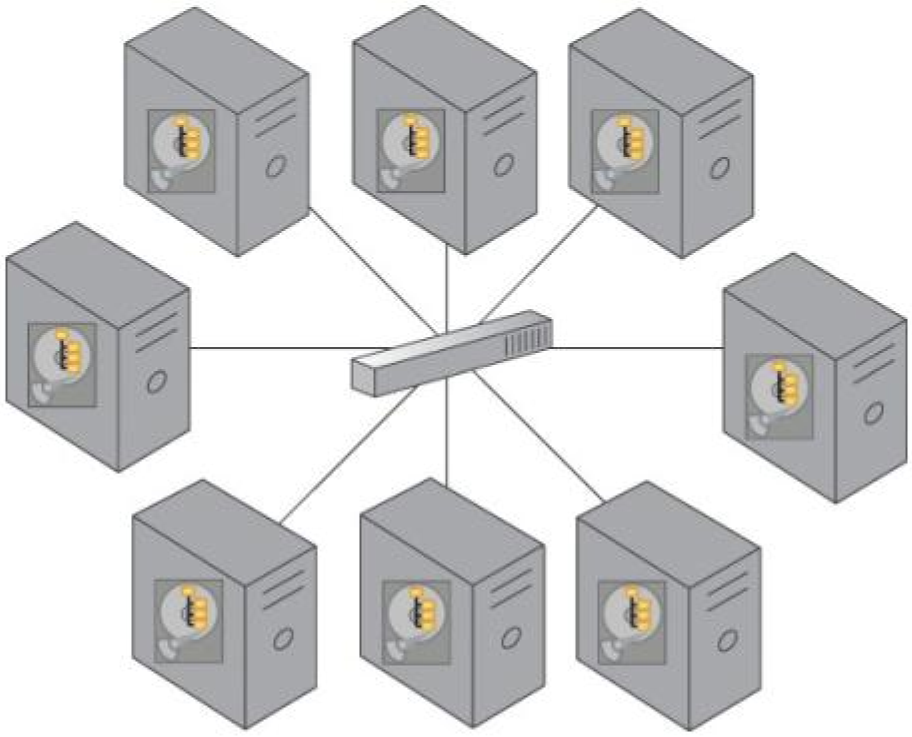


Figure 3.3 The symbol used to represent distributed file systems [26]

3.1.3 NoSQL [26]

A Not-only SQL (NoSQL) database is a non-relational database that is highly scalable, fault-tolerant and specifically designed to house semi-structured and unstructured data. A NoSQL database often provides an API-based query interface that can be called from within an application. NoSQL databases also support query languages other than Structured Query Language (SQL) because SQL was designed to query structured data stored within a relational database. As an example, a NoSQL database that is optimized to store XML files will often use XQuery as the query language. Likewise, a NoSQL database designed to store RDF data will use SPARQL to query the relationships it contains. That being said, there are some NoSQL databases that also provide an SQL-like query interface, as shown in Figure 3.4.

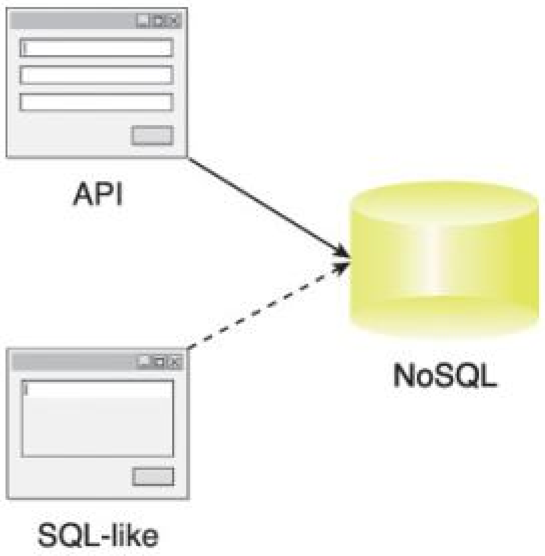
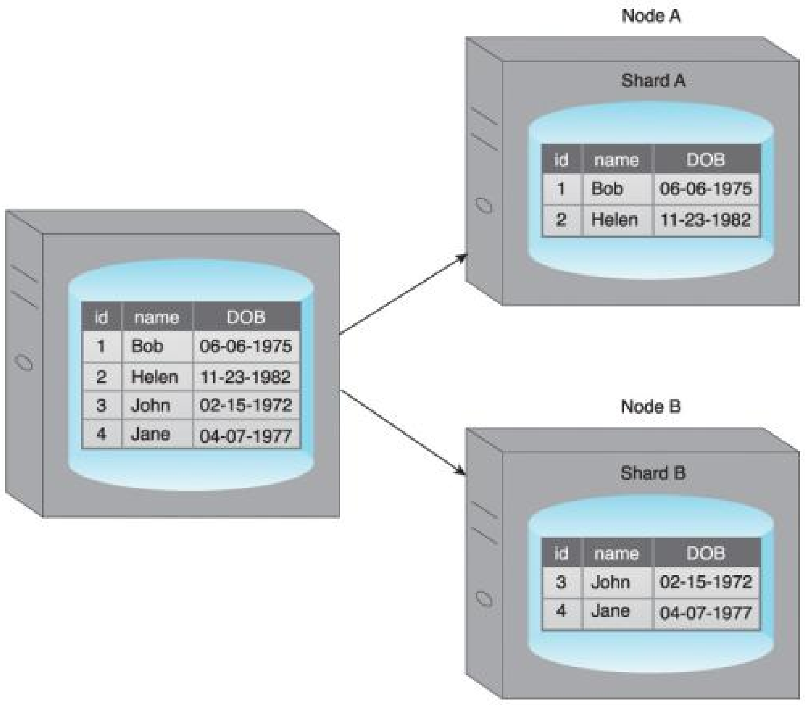


Figure 3.4 A NoSQL database can provide an API- or SQL-like query interface [26]

3.1.4 Sharding [26]

Sharding is the process of horizontally partitioning a large dataset into a collection of smaller, more manageable datasets called shards. The shards are distributed across multiple nodes, where a node is a server or a machine (Figure 3.5). Each shard is stored on a separate node and each node is responsible for only the data stored on it. Each shard shares the same schema, and all shards collectively represent the complete dataset.



id	name	DOB
1	Bob	06-06-1975
2	Helen	11-23-1982
3	John	02-15-1972
4	Jane	04-07-1977

id	name	DOB
1	Bob	06-06-1975
2	Helen	11-23-1982

id	name	DOB
3	John	02-15-1972
4	Jane	04-07-1977

Figure 3.5 An example of sharding where a dataset is spread across Node A and Node B, resulting in Shard A and Shard B, respectively. [26]

Sharding is often transparent to the client, but this is not a requirement. Sharding allows the distribution of processing loads across multiple nodes to achieve horizontal scalability. Horizontal scaling is a method for increasing a system's capacity by adding similar or higher capacity resources alongside existing resources. Since each node is responsible for only a part of the whole dataset, read/write times are greatly improved.
Figure 3.6 presents an illustration of how sharding works in practice:

1. Each shard can independently service reads and writes for the specific subset of data that it is responsible for.
2. Depending on the query, data may need to be fetched from both shards.

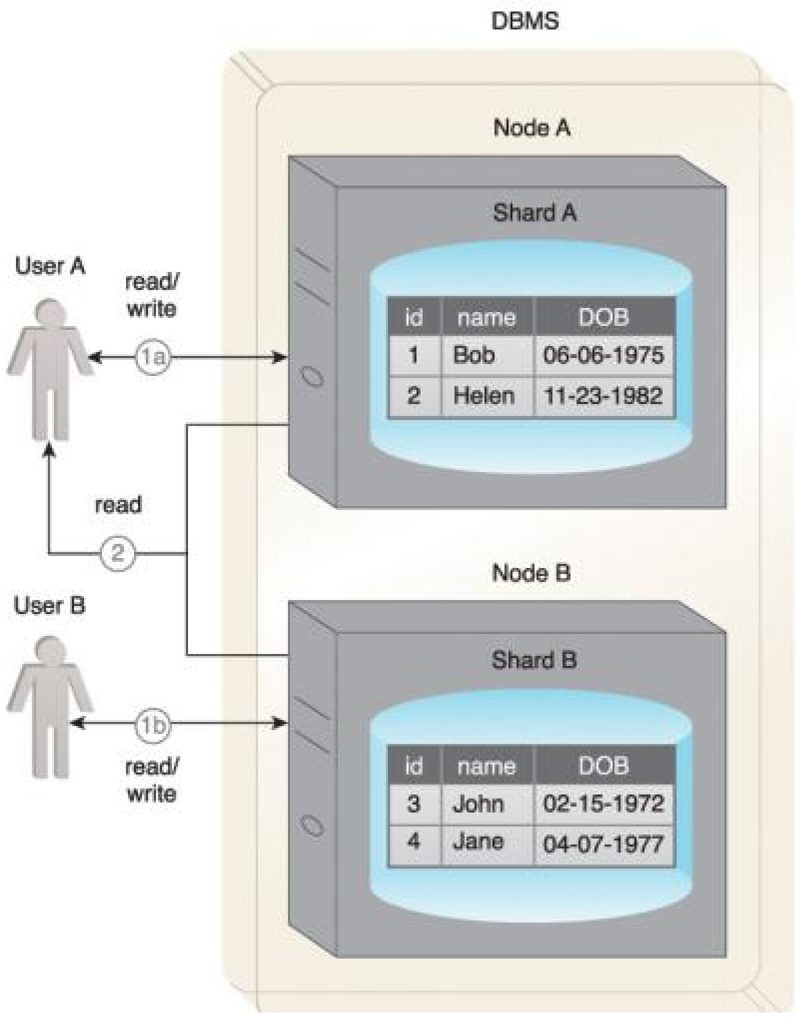


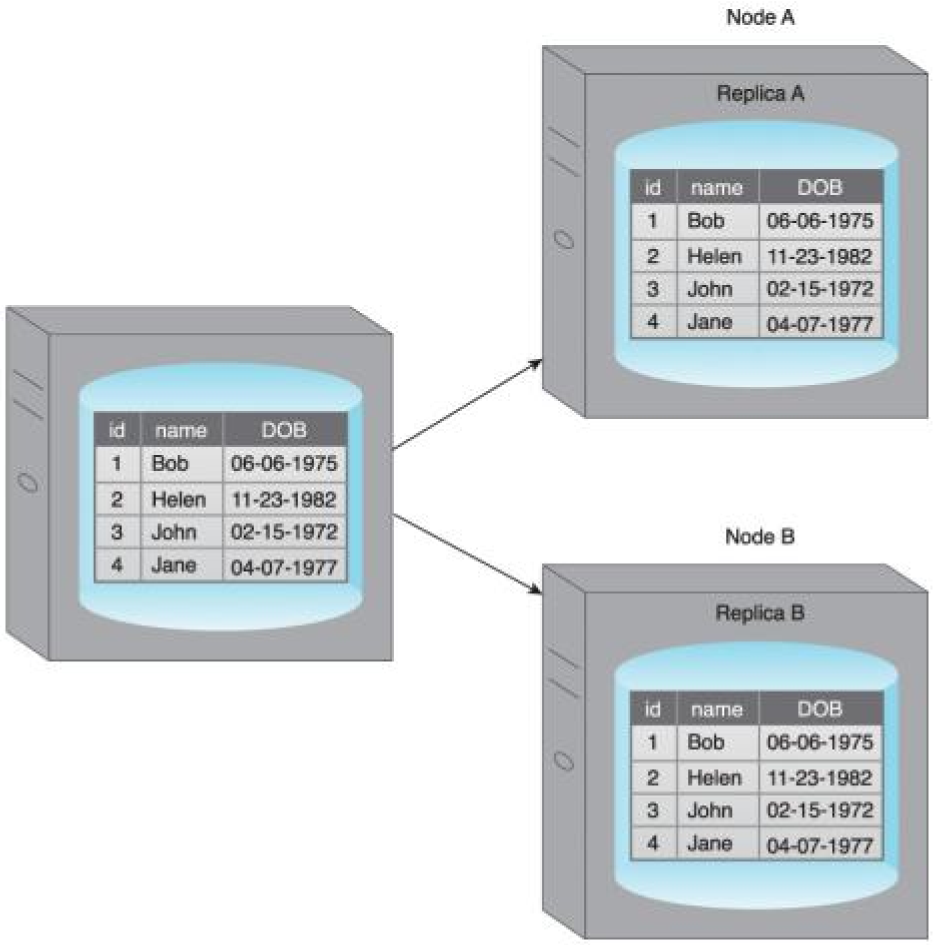
Figure 3.6 A sharding example where data is fetched from both Node A and Node B [26]

A benefit of sharding is that it provides partial tolerance toward failures. In case of a node failure, only data stored on that node is affected. With regards to data partitioning, query patterns need to be taken into account so that shards themselves do not become performance bottlenecks. For example, queries requiring data from multiple shards will impose performance penalties. Data locality keeps commonly accessed data co-located on a single shard and helps counter such performance issues.

3.1.5 Replication [26]

Replication stores multiple copies of a dataset, known as replicas, on multiple nodes (Figure 3.7). Replication provides scalability and availability due to the fact that the same data is replicated on various nodes. Fault tolerance is also achieved since data redundancy ensures that data is not lost when an individual node fails. There are two different methods that are used to implement replication:

• master-slave
• peer-to-peer



id	name	DOB
1	Bob	06-06-1975
2	Helen	11-23-1982
3	John	02-15-1972
4	Jane	04-07-1977

id	name	DOB
1	Bob	06-06-1975
2	Helen	11-23-1982
3	John	02-15-1972
4	Jane	04-07-1977

id	name	DOB
1	Bob	06-06-1975
2	Helen	11-23-1982
3	John	02-15-1972
4	Jane	04-07-1977

Figure 3.7 An example of replication where a dataset is replicated to Node A and Node B, resulting in Replica A and Replica B [26]

Master-Slave
During master-slave replication, nodes are arranged in a master-slave configuration, and all data is written to a master node. Once saved, the data is replicated over to multiple slave nodes. All external write requests, including insert, update and delete, occur on the master node, whereas read requests can be fulfilled by any slave node. In Figure 3.8, writes are managed by the master node and data can be read from either Slave A or Slave B.

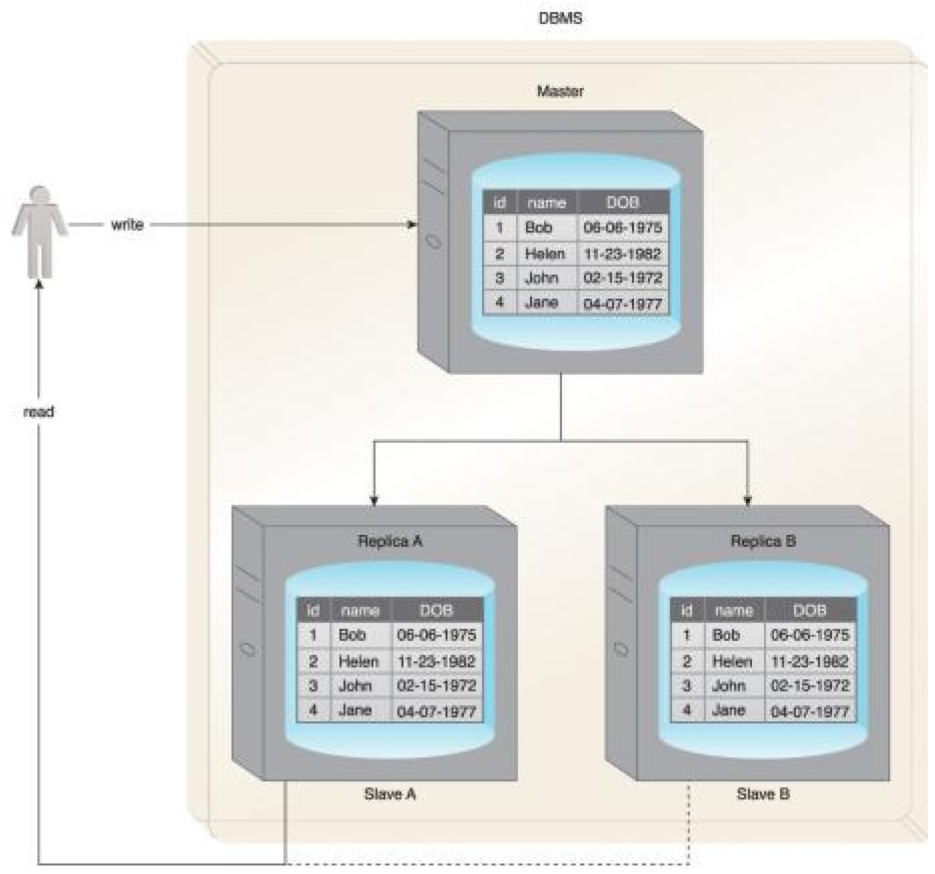


Figure 3.8 An example of master-slave replication where Master A is the single point of contact for all writes, and data can be read from Slave A and Slave B [26]

Master-slave replication is ideal for read intensive loads rather than write intensive loads since growing read demands can be managed by horizontal scaling to add more slave nodes. Writes are consistent, as all writes are coordinated by the master node. The implication is that write performance will suffer as the amount of writes increases. If the master node fails, reads are still possible via any of the slave nodes.

A slave node can be configured as a backup node for the master node. In the event that the master node fails, writes are not supported until a master node is reestablished. The master node is either resurrected from a backup of the master node, or a new master node is chosen from the slave nodes.

One concern with master-slave replication is read inconsistency, which can be an issue if a slave node is read prior to an update to the master being copied to it. To ensure read consistency, a voting system can be implemented where a read is declared consistent if the majority of the slaves contain the same version of the record. Implementation of such a voting system requires a reliable and fast communication mechanism between the slaves.

Figure 3.9 illustrates a scenario where read inconsistency occurs.

1. User A updates data.

2. The data is copied over to Slave A by the Master.
3. Before the data is copied over to Slave B, User B tries to read the data from Slave B, which results in an inconsistent read.
4. The data will eventually become consistent when Slave B is updated by the Master.

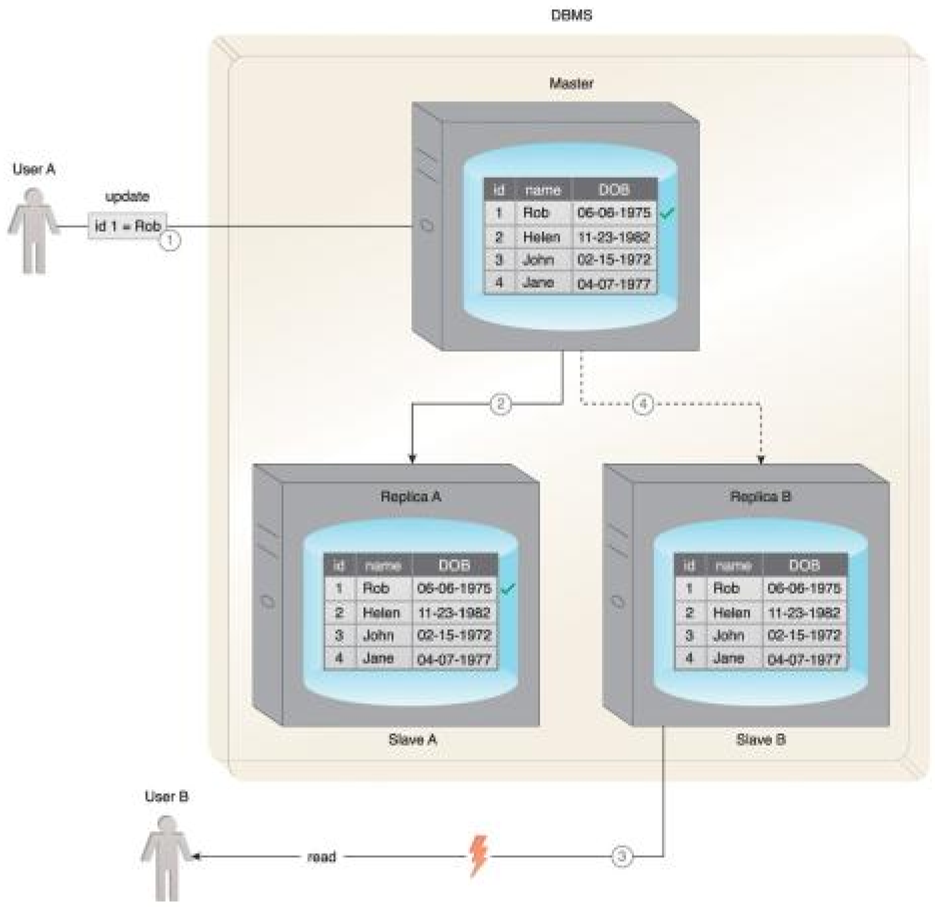


Figure 3.9 An example of master-slave replication where read inconsistency occurs. [26]

Peer-to-Peer

With peer-to-peer replication, all nodes operate at the same level. In other words, there is not a master-slave relationship between the nodes. Each node, known as a peer, is equally capable of handling reads and writes. Each write is copied to all peers, as illustrated in Figure 3.10.

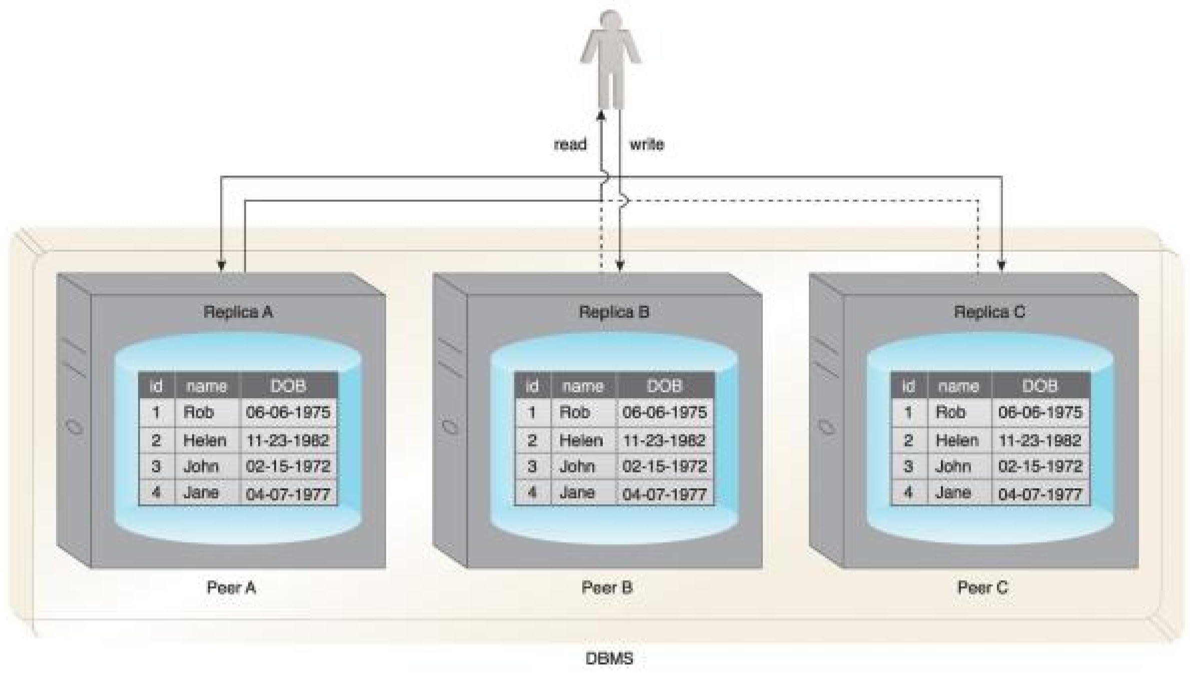


Figure 3.10 Writes are copied to Peers A, B and C simultaneously. Data is read from Peer A, but it can also be read from Peers B or C. [26]

Peer-to-peer replication is prone to write inconsistencies that occur as a result of a simultaneous update of the same data across multiple peers. This can be addressed by implementing either a pessimistic or optimistic concurrency strategy.

- Pessimistic concurrency is a proactive strategy that prevents inconsistency. It uses locking to ensure that only one update to a record can occur at a time. However, this is detrimental to availability since the database record being updated remains unavailable until all locks are released.
- Optimistic concurrency is a reactive strategy that does not use locking. Instead, it allows inconsistency to occur with knowledge that eventually consistency will be achieved after all updates have propagated.

With optimistic concurrency, peers may remain inconsistent for some period of time before attaining consistency. However, the database remains available as no locking is involved. Like master-slave replication, reads can be inconsistent during the time period when some of the peers have completed their updates while others perform their updates. However, reads eventually become consistent when the updates have been executed on all peers.

To ensure read consistency, a voting system can be implemented where a read is declared consistent if the majority of the peers contain the same version of the record. As previously indicated, implementation of such a voting system requires a reliable and fast communication mechanism between the peers.

Figure 3.11 demonstrates a scenario where an inconsistent read occurs.

1. User A updates data.
2. a. The data is copied over to Peer A.
   b. The data is copied over to Peer B.
3. Before the data is copied over to Peer C, User B tries to read the data from Peer C, resulting in an inconsistent read.
4. The data will eventually be updated on Peer C, and the database will once again become consistent.

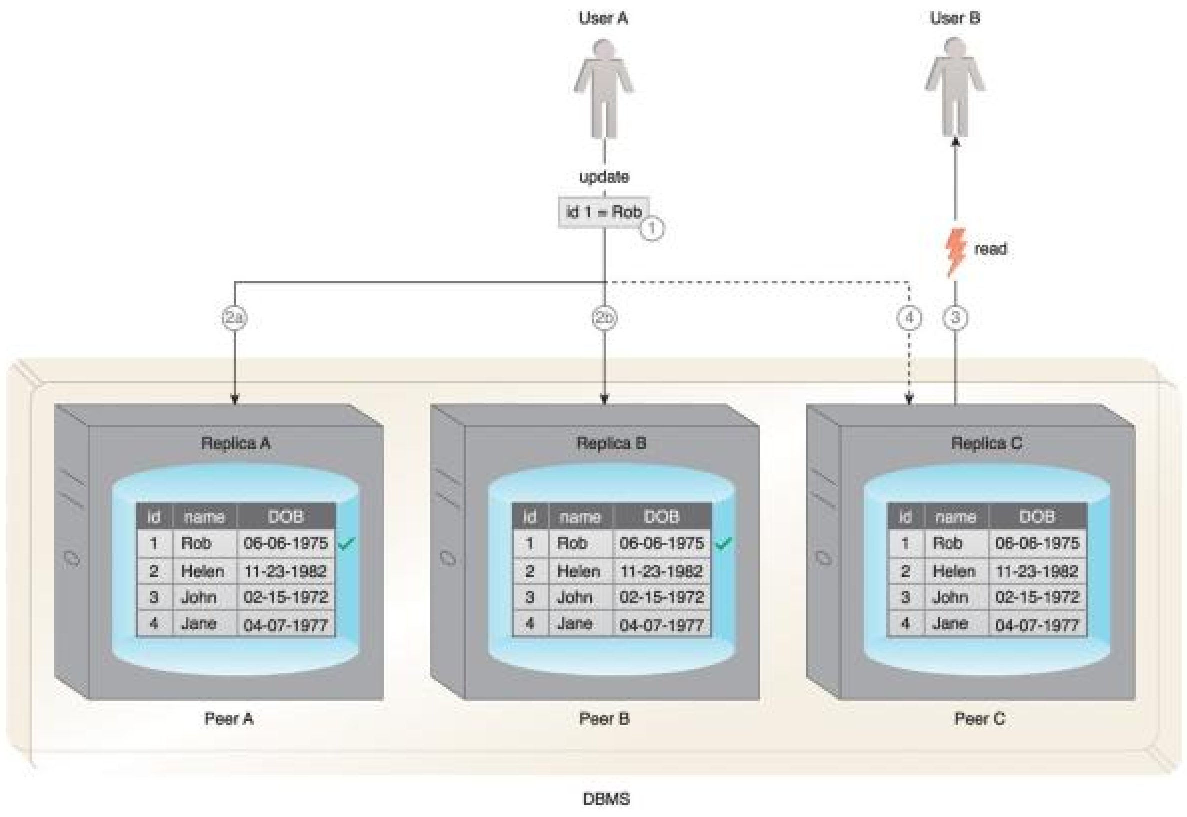


Figure 3.11 An example of peer-to-peer replication where an inconsistent read occurs. [26]

3.1.6 Sharding and Replication [26]

To improve on the limited fault tolerance offered by sharding, while additionally benefiting from the increased availability and scalability of replication, both sharding and replication can be combined, as shown in Figure 3.12.

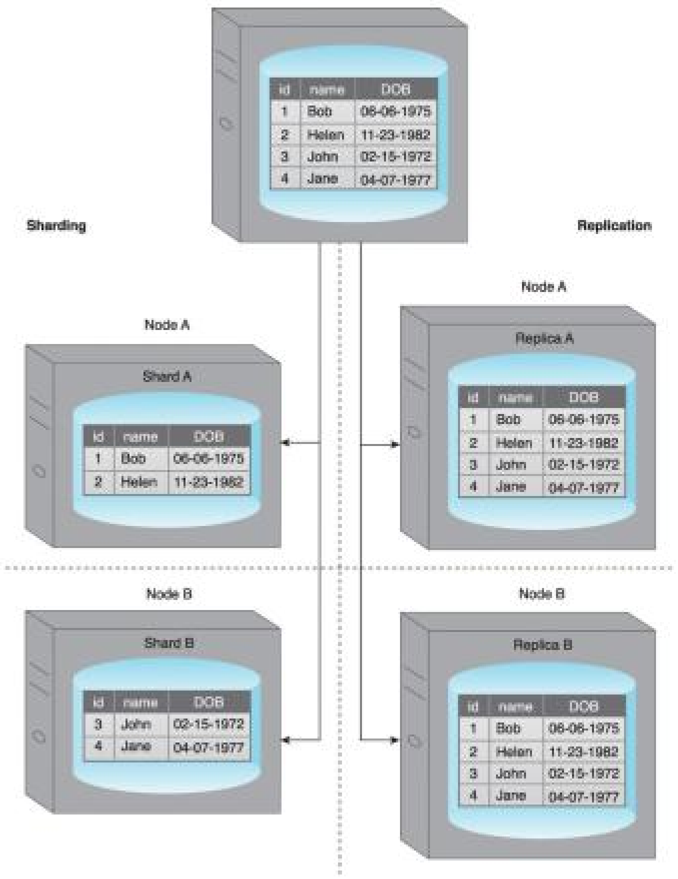


Figure 3.12 A comparison of sharding and replication that shows how a dataset is distributed between two nodes with the different approaches. [26]

This section covers the following combinations:

- sharding and master-slave replication
- sharding and peer-to-peer replication

Combining Sharding and Master-Slave Replication

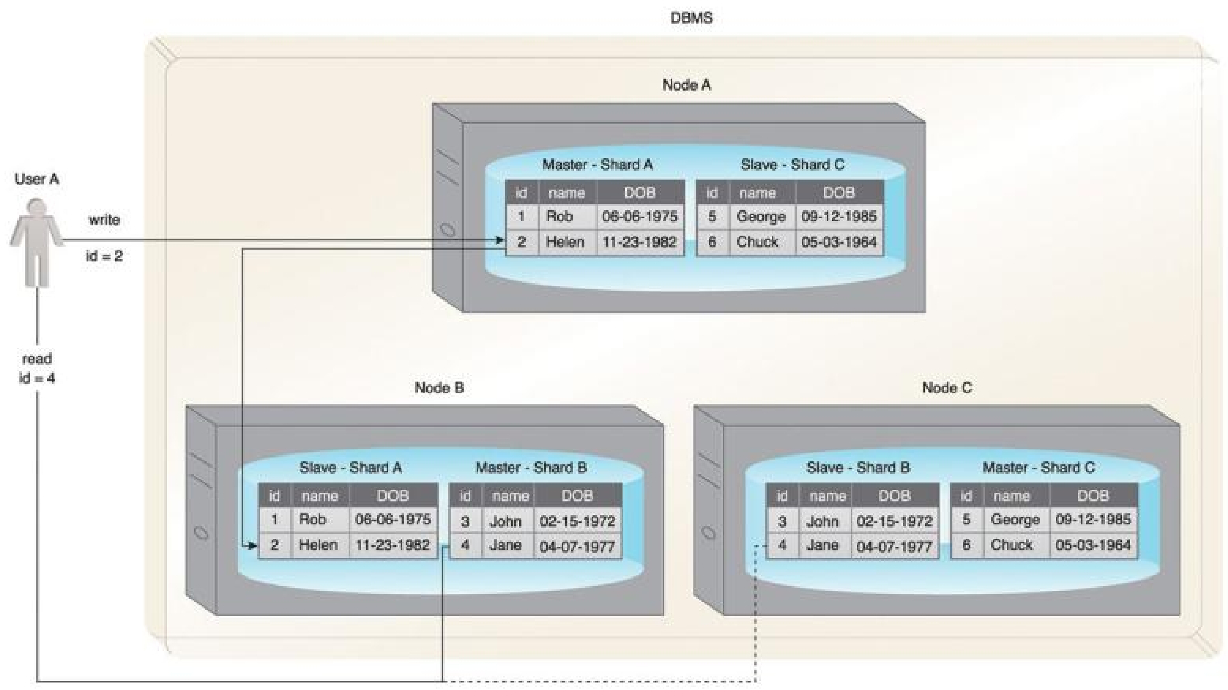
When sharding is combined with master-slave replication, multiple shards become slaves of a single master, and the master itself is a shard. Although this results in multiple masters, a single slave-shard can only be managed by a single master-shard.

Write consistency is maintained by the master-shard. However, if the master-shard becomes non-operational or a network outage occurs, fault tolerance with regards to write operations is impacted. Replicas of shards are kept on multiple slave nodes to provide scalability and fault tolerance for read operations.

In Figure 3.13:

- Each node acts both as a master and a slave for different shards.
- Writes (id = 2) to Shard A are regulated by Node A, as it is the master for Shard A.
- Node A replicates data (id = 2) to Node B, which is a slave for Shard A.

• Reads (id = 4) can be served directly by either Node B or Node C as they each contain Shard B.



id	name	DOB
1	Rob	06-06-1975
2	Helen	11-23-1982

id	name	DOB
5	George	09-12-1985
6	Chuck	05-03-1964

id	name	DOB
3	John	02-15-1972
4	Jane	04-07-1977

Figure 3.13 A comparison of sharding and replication that shows how a dataset is distributed between two nodes with the different approaches. [26]

Combining Sharding and Peer-to-Peer Replication

When combining sharding with peer-to-peer replication, each shard is replicated to multiple peers, and each peer is only responsible for a subset of the overall dataset.

Collectively, this helps achieve increased scalability and fault tolerance. As there is no master involved, there is no single point of failure and fault-tolerance for both read and write operations is supported.

In Figure 3.14:

• Each node contains replicas of two different shards.

• Writes (id = 3) are replicated to both Node A and Node C (Peers) as they are responsible for Shard C.

• Reads (id = 6) can be served by either Node B or Node C as they each contain Shard B.

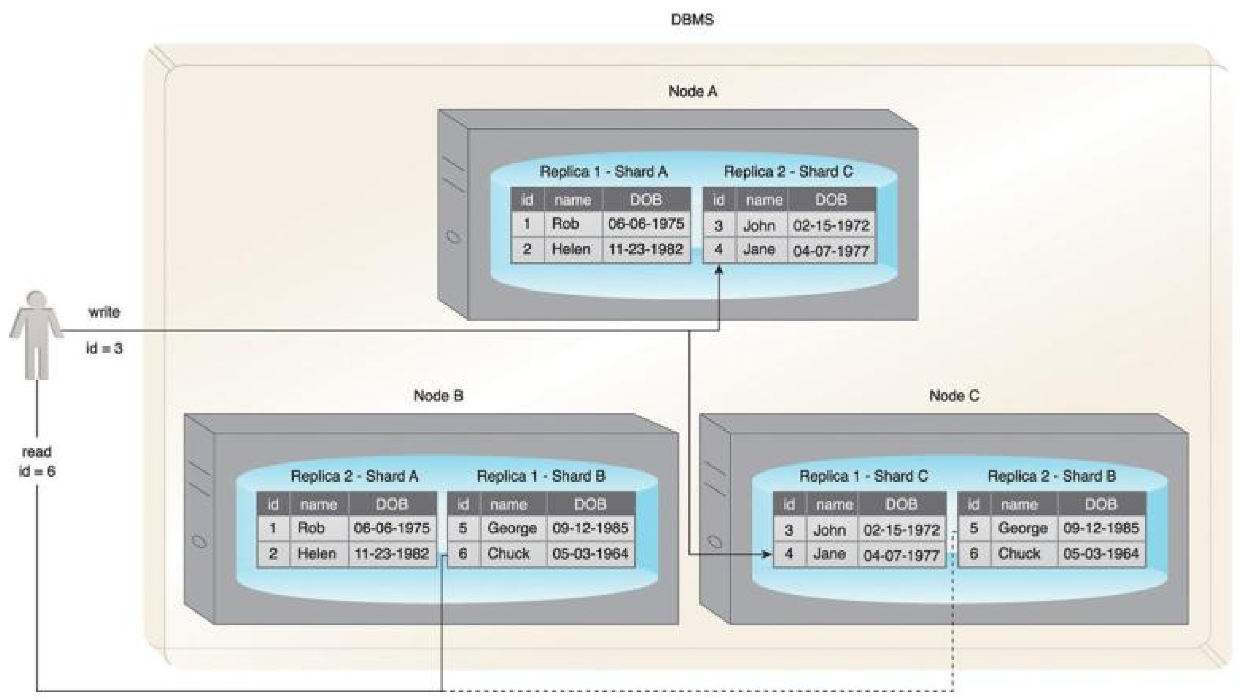


Figure 3.14 An example of the combination of sharding and peer-to-peer replication [26]

3.1.7 CAP Theorem [26]

The Consistency, Availability, and Partition tolerance (CAP) theorem, also known as Brewer's theorem, expresses a triple constraint related to distributed database systems. It states that a distributed database system, running on a cluster, can only provide two of the following three properties:

• ***Consistency*** – A read from any node results in the same data across multiple nodes (Figure 3.15)

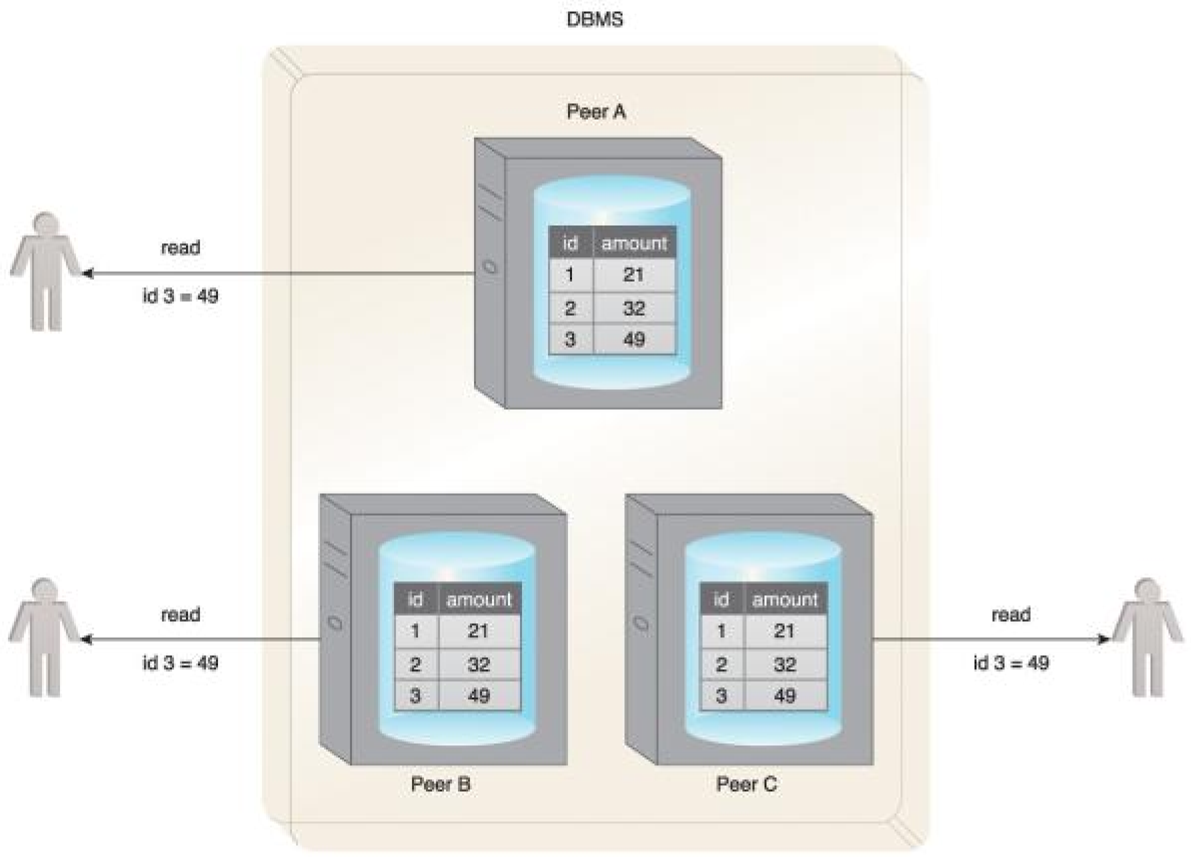


Figure 3.15 Consistency: all three users get the same value for the amount column even though three different nodes are serving the record. [26]

• ***Availability*** – A read/write request will always be acknowledged in the form of a success or a failure (Figure 3.16).

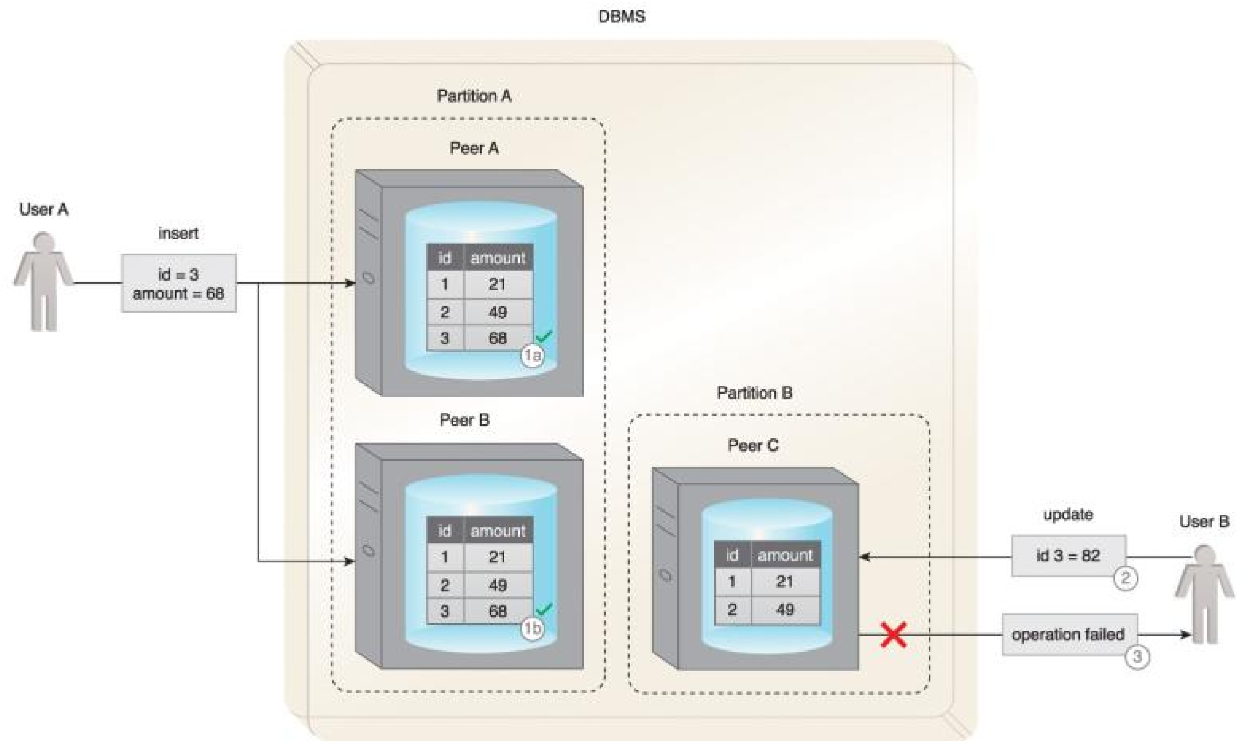


Figure 3.16 Availability and partition tolerance: in the event of a communication failure, requests from both users are still serviced (1, 2). However, with User B, the update fails as the record with id = 3 has not been copied over to Peer C. The user is duly notified (3) that the update has failed. [26]

• ***Partition tolerance*** – The database system can tolerate communication outages that split the cluster into multiple silos and can still service read/write requests (Figure 3.16).
The following scenarios demonstrate why only two of the three properties of the CAP theorem are simultaneously supportable. To aid this discussion, Figure 3.17 provides a Venn diagram showing the areas of overlap between consistency, availability and partition tolerance.

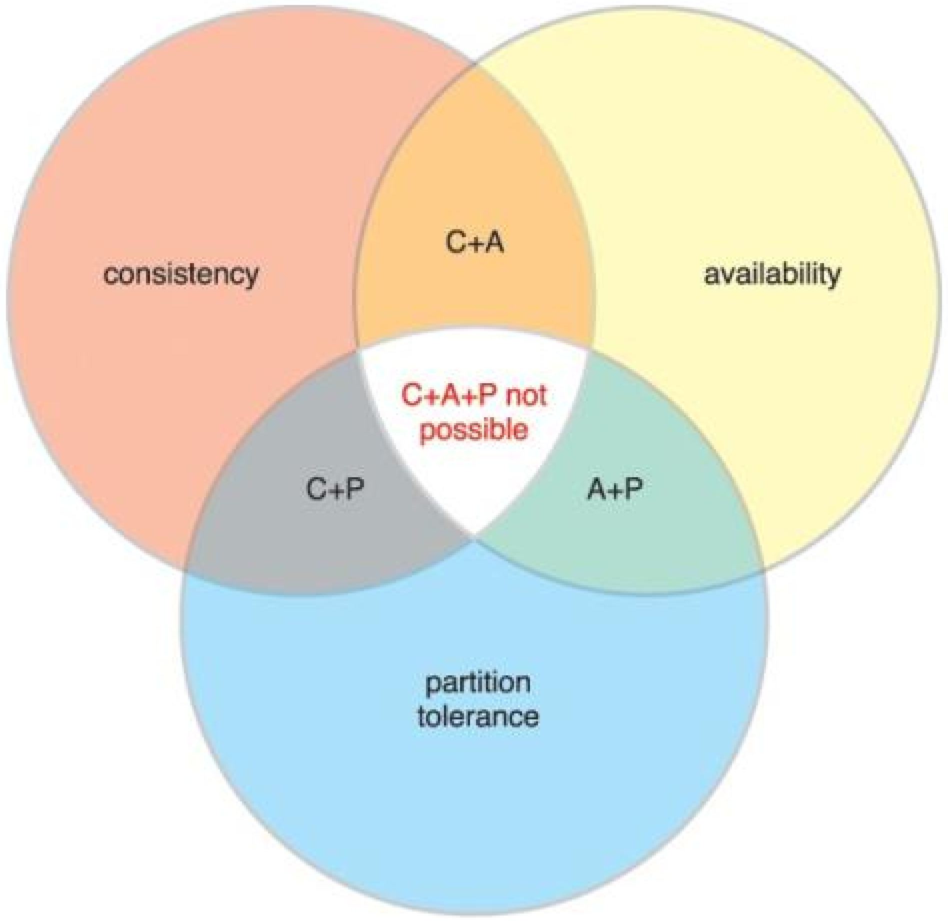


Figure 3.17 A Venn diagram summarizing the CAP theorem. [26]

If consistency (C) and availability (A) are required, available nodes need to communicate to ensure consistency (C). Therefore, partition tolerance (P) is not possible.

If consistency (C) and partition tolerance (P) are required, nodes cannot remain available (A) as the nodes will become unavailable while achieving a state of consistency (C).

If availability (A) and partition tolerance (P) are required, then consistency (C) is not possible because of the data communication requirement between the nodes. So, the database can remain available (A) but with inconsistent results.

In a distributed database, scalability and fault tolerance can be improved through additional nodes, although this challenges consistency (C). The addition of nodes can also cause availability (A) to suffer due to the latency caused by increased communication between nodes.

Distributed database systems cannot be 100% partition tolerant (P). Although communication outages are rare and temporary, partition tolerance (P) must always be supported by a distributed database; therefore, CAP is generally a choice between choosing either C+P or A+P. The requirements of the system will dictate which is chosen.

3.1.8 ACID [26]

ACID is a database design principle related to transaction management. It is an acronym that stands for:

- atomicity
- consistency
- isolation
- durability

ACID is a transaction management style that leverages pessimistic concurrency controls to ensure consistency is maintained through the application of record locks. ACID is the traditional approach to database transaction management as it is leveraged by relational database management systems.

Atomicity ensures that all operations will always succeed or fail completely. In other words, there are no partial transactions.

The following steps are illustrated in Figure 3.18:

1. A user attempts to update three records as a part of a transaction.
2. Two records are successfully updated before the occurrence of an error.
3. As a result, the database roll backs any partial effects of the transaction and puts the system back to its prior state.

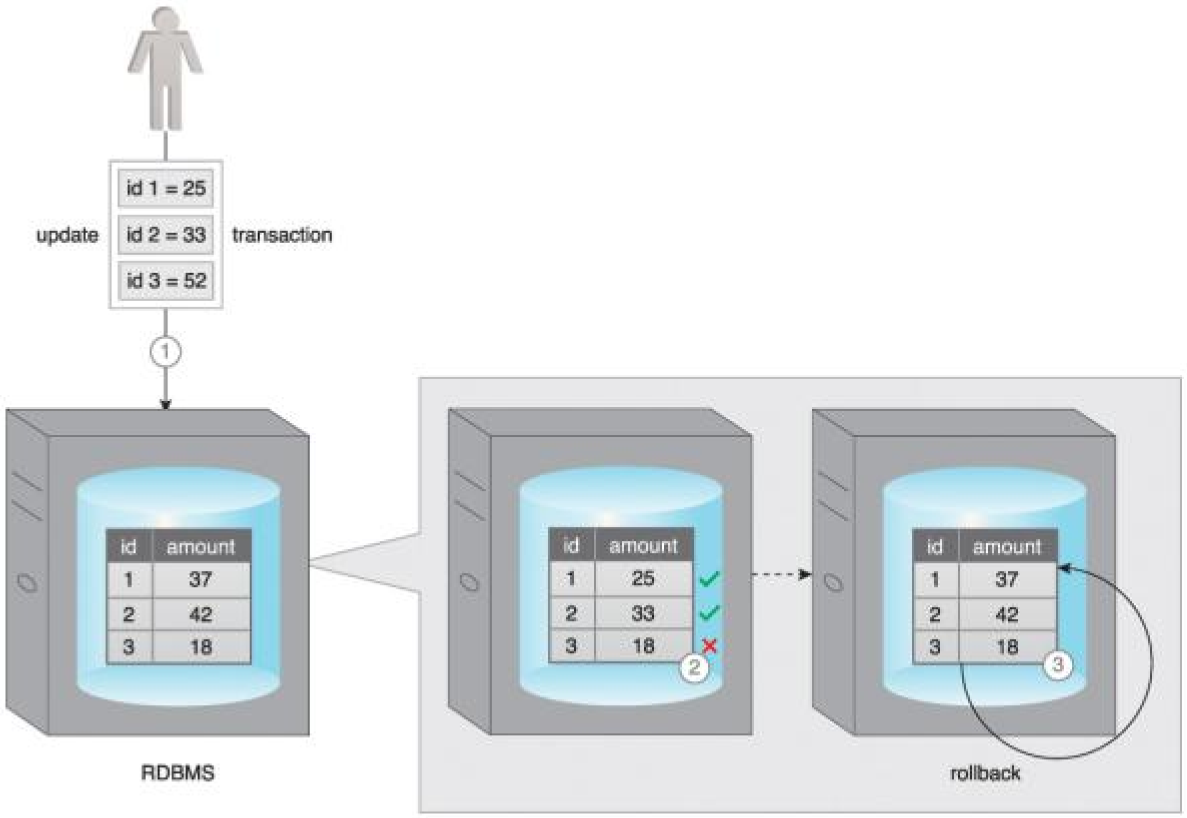


Figure 3.18 An example of the atomicity property of ACID is evident here. [26]

Consistency ensures that the database will always remain in a consistent state by ensuring that only data that conforms to the constraints of the database schema can be written to the database. Thus a database that is in a consistent state will remain in a consistent state following a successful transaction.

In Figure 3.19:

1. A user attempts to update the amount column of the table that is of type float with a varchar value.
2. The database applies its validation check and rejects this update because the value violates the constraint checks for the amount column.

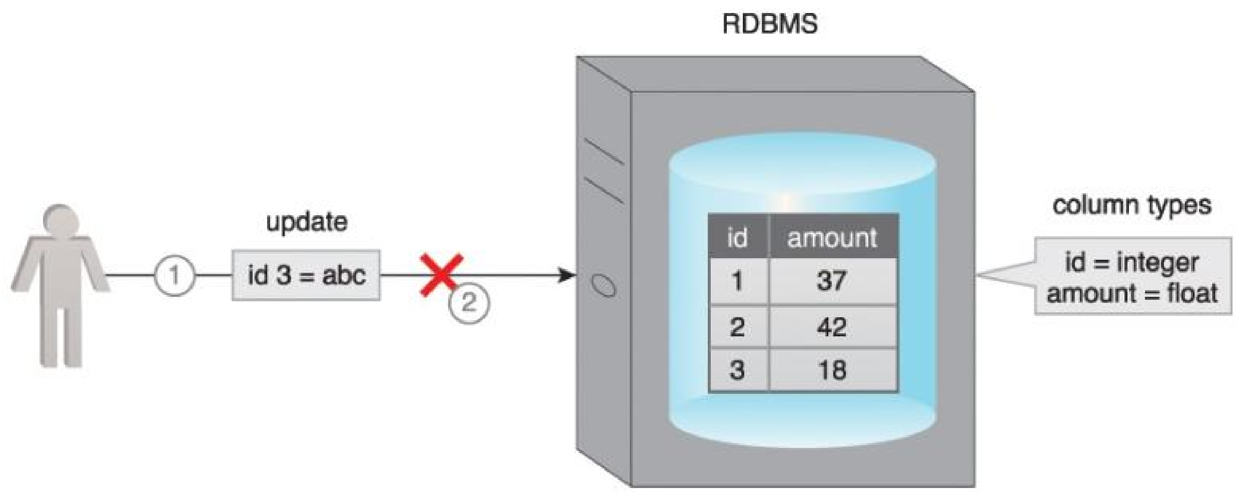


Figure 3.19 An example of the consistency of ACID [26]

Isolation ensures that the results of a transaction are not visible to other operations until it is complete.

In Figure 3.20:

1. User A attempts to update two records as part of a transaction.
2. The database successfully updates the first record.
3. However, before it can update the second record, User B attempts to update the same record. The database does not permit User B's update until User A's update succeeds or fails in full. This occurs because the record with id3 is locked by the database until the transaction is complete.

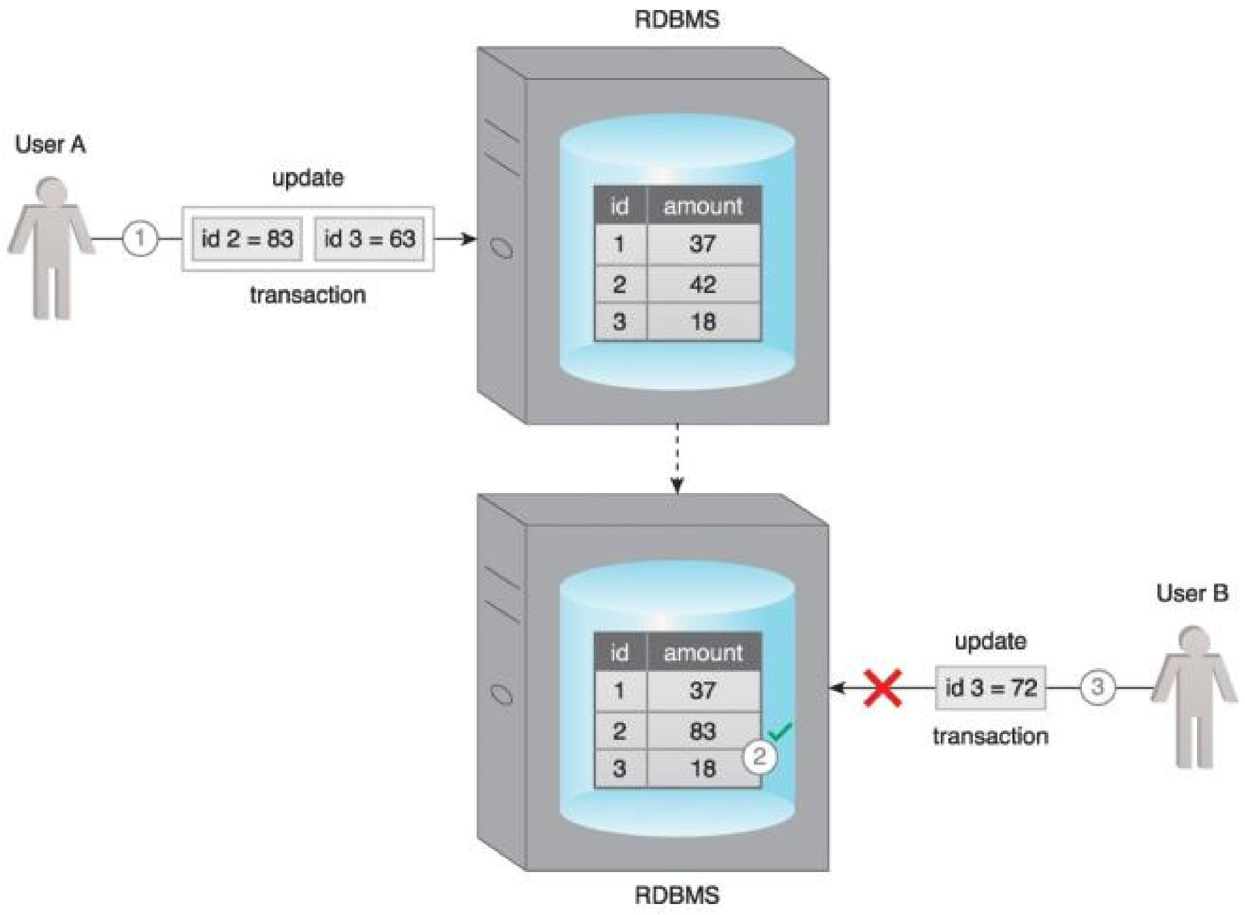


Figure 3.20 An example of the isolation property of ACID [26]

Durability ensures that the results of an operation are permanent. In other words, once a transaction has been committed, it cannot be rolled back. This is irrespective of any system failure.

In Figure 3.21:

1. A user updates a record as part of a transaction.
2. The database successfully updates the record.
3. Right after this update, a power failure occurs. The database maintains its state while there is no power.
4. The power is resumed.
5. The database serves the record as per last update when requested by the user.

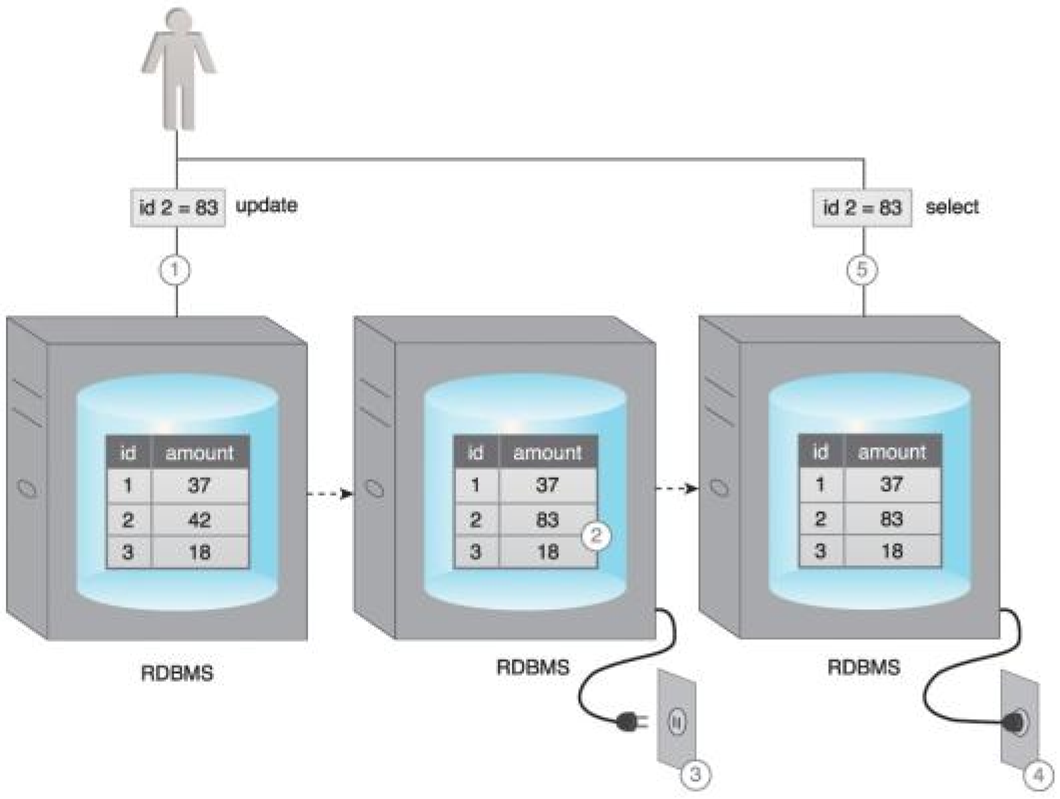


Figure 3.21 The durability characteristic of ACID. [26]

Figure 3.22 shows the results of the application of the ACID principle:

1. User A attempts to update a record as part of a transaction.
2. The database validates the value and the update is successfully applied.
3. After the successful completion of the transaction, when Users B and C request the same record, the database provides the updated value to both the users.

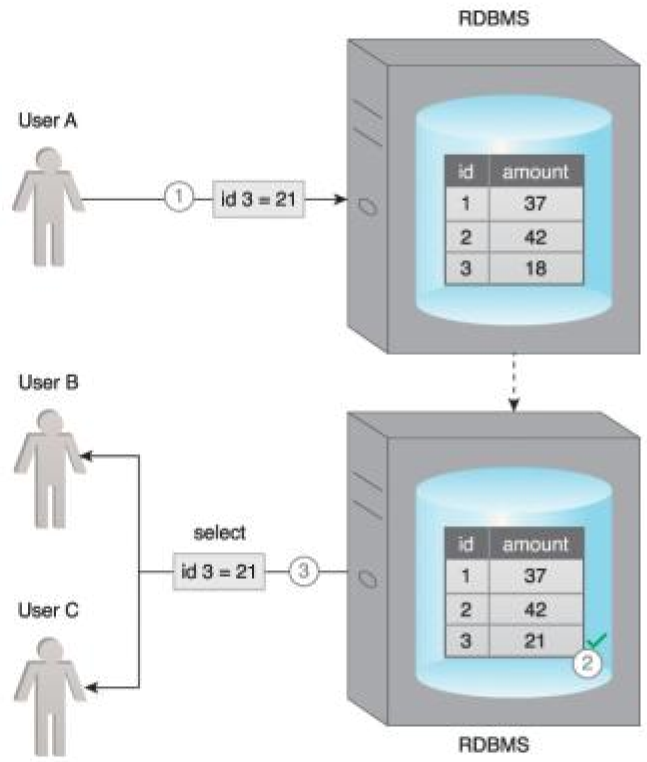


Figure 3.22 The ACID principle results in consistent database behavior. [26]

3.1.9 BASE [26]

BASE is a database design principle based on the CAP theorem and leveraged by database systems that use distributed technology. BASE stands for:
• basically available
• soft state
• eventual consistency

When a database supports BASE, it favors availability over consistency. In other words, the database is A+P from a CAP perspective. In essence, BASE leverages optimistic concurrency by relaxing the strong consistency constraints mandated by the ACID properties.
If a database is "basically available," that database will always acknowledge a client's request, either in the form of the requested data or a success/failure notification.
In Figure 3.23, the database is basically available, even though it has been partitioned as a result of a network failure.

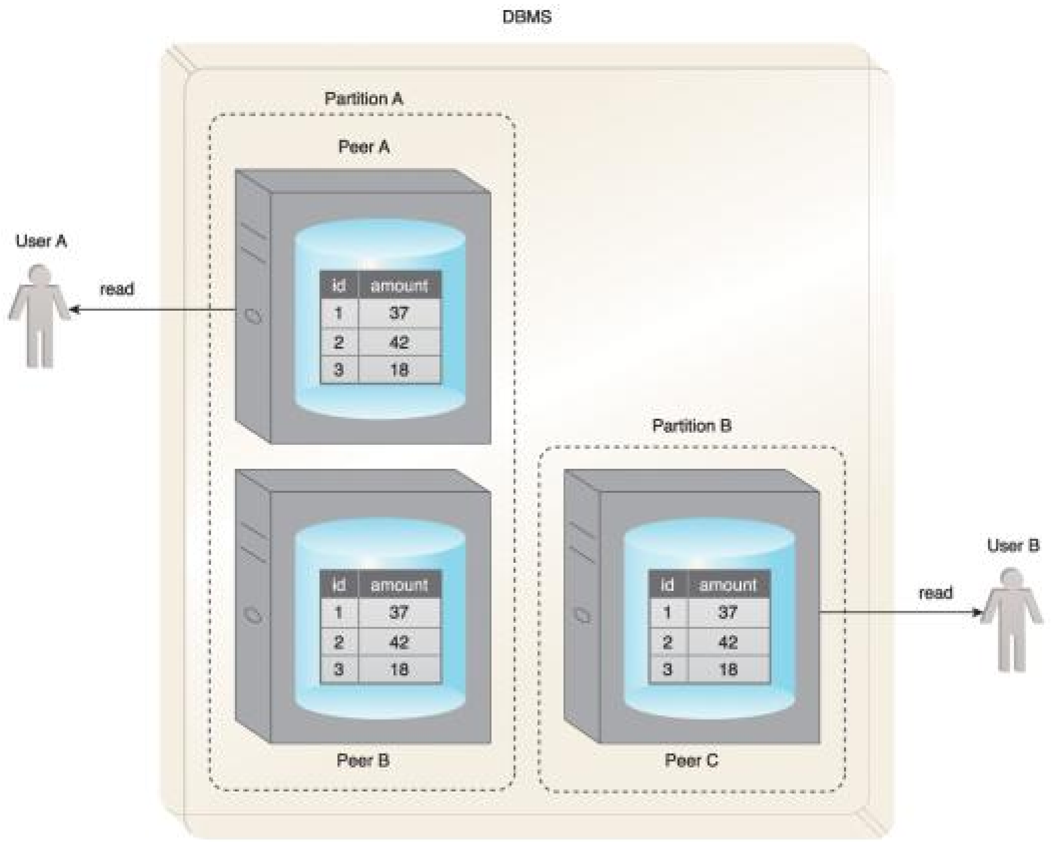


Figure 3.23 User A and User B receive data despite the database being partitioned by a network failure. [26]

Soft state means that a database may be in an inconsistent state when data is read; thus, the results may change if the same data is requested again. This is because the data could be updated for consistency, even though no user has written to the database between the two reads. This property is closely related to eventual consistency.

In Figure 3.24:

1. User A updates a record on Peer A.
2. Before the other peers are updated, User B requests the same record from Peer C.
3. The database is now in a soft state, and stale data is returned to User B.

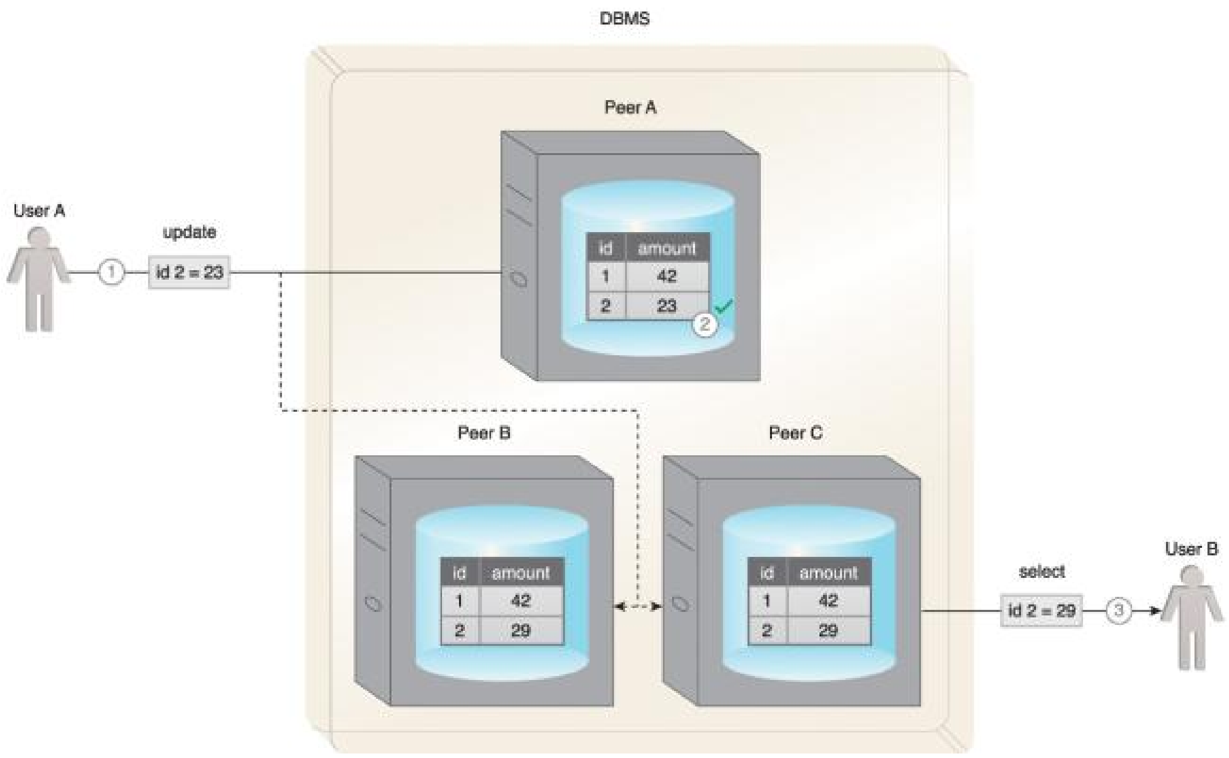


Figure 3.24 An example of the soft state property of BASE is shown here. [26]

Eventual consistency is the state in which reads by different clients, immediately following a write to the database, may not return consistent results. The database only attains consistency once the changes have been propagated to all nodes. While the database is in the process of attaining the state of eventual consistency, it will be in a soft state.

In Figure 3.25:

1. User A updates a record.
2. The record only gets updated at Peer A, but before the other peers can be updated, User B requests the same record.
3. The database is now in a soft state. Stale data is returned to User B from Peer C.
4. However, the consistency is eventually attained, and User C gets the correct value.

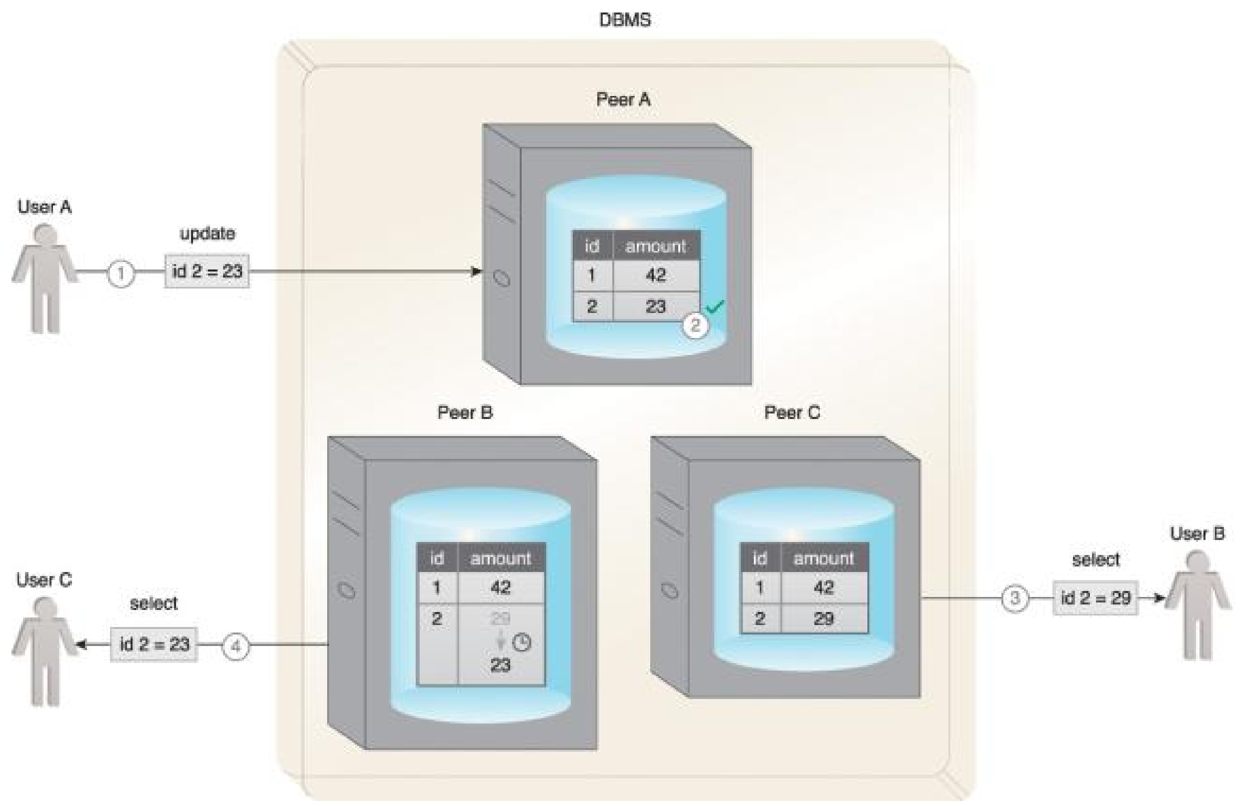


Figure 3.25 An example of the eventual consistency property of BASE. [26]

BASE emphasizes availability over immediate consistency, in contrast to ACID, which ensures immediate consistency at the expense of availability due to record locking. This soft approach toward consistency allows BASE compliant databases to serve multiple clients without any latency albeit serving inconsistent results. However, BASE-compliant databases are not useful for transactional systems where lack of consistency is a concern.

3.1.10 Case Study Example [26]

ETI's IT environment currently utilizes both Linux and Windows operating systems. Consequently, both ext and NTFS file systems are in use. The webserversand some of the application servers employ ext, while the rest of the application servers, the database servers and the end-users' PCs are configured to use NTFS. Network-attached storage (NAS) configured with RAID 5 is also used for fault tolerant document storage. Although the IT team is conversant with file systems, the concepts of cluster, distributed file system and NoSQL are new to the group. Nevertheless, after a discussion with the trained IT team members, the entire group is able to understand these concepts and technologies.
ETI's current IT landscape comprises entirely of relational databases that employ the ACID database design principle. The IT team has no understanding of the BASE principle and is having trouble comprehending the CAP theorem. Some of the team

members are unsure about the need and importance of these concepts with regards to Big Data dataset storage. Seeing this, the IT-trained employees try to ease their fellow team members' confusion by explaining that these concepts are only applicable to the storage of enormous amounts of data in a distributed fashion on a cluster. Clusters have become the obvious choice for storing very large volume of data due to their ability to support linear scalability by scaling out.

Since clusters are comprised of nodes connected via a network, communication failures that create silos or partitions of a cluster are inevitable. To address the partition issue, the BASE principle and CAP theorem are introduced. They further explain that any database following the BASE principle becomes more responsive to its clients, albeit the data being read may be inconsistent when compared to a database that follows the ACID principle. Having understood the BASE principle, the IT team more easily comprehends why a database implemented in a cluster has to choose between consistency and availability.

Although none of the existing relational databases use sharding, almost all relational databases are replicated for disaster recovery and operational reporting. To better understand the concepts of sharding and replication, the IT team goes through an exercise of how these concepts can be applied to the insurance quote data as a large number of quotes are created and accessed quickly. For sharding, the team believes that using the type (the insurance sector—heath, building, marine and aviation) of the insurance quote as sharding criteria will create a balanced set of data across multiple nodes, for queries are mostly executed within the same insurance sector, and inter-sector queries are rare. With regards to replication, the team is in favor of choosing a NoSQL database that implements the peer-to-peer replication strategy. The reason behind their decision is that the insurance quotes are created and retrieved quite frequently but seldom updated. Hence the chances of getting an inconsistent record are low. Considering this, the team favors read/write performance over consistency by choosing peer-to-peer replication.

3.2 APACHE MAHOUT [27, 30]

Mahout is an open source *machine learning* library from Apache. The algorithms it implements fall under the broad umbrella of machine learning or collective intelligence. This can mean many things, but at the moment for Mahout it means primarily recommender engines (collaborative filtering), clustering, and classification.

It's also *scalable*. Mahout aims to be the machine learning tool of choice when the collection of data to be processed is very large, perhaps far too large for a single machine. In its current incarnation, these scalable machine learning implementations in Mahout are written in Java, and some portions are built upon Apache's Hadoop distributed computation project.

Finally, it's a Java library. It doesn't provide a user interface, a prepackaged server, or an installer. It's a framework of tools intended to be used and adapted by developers.
To set the stage, this chapter will take a brief look at the sorts of machine learning that Mahout can help you perform on your data—using recommender engines, clustering, and classification—by looking at some familiar real-world instances.

Features of Mahout [30]
The primitive features of Apache Mahout are listed below.

- The algorithms of Mahout are written on top of Hadoop, so it works well in distributed environment. Mahout uses the Apache Hadoop library to scale effectively in the cloud.
- Mahout offers the coder a ready-to-use framework for doing data mining tasks on large volumes of data.
- Mahout lets applications to analyze large sets of data effectively and in quick time.
- Includes several MapReduce enabled clustering implementations such as k-means, fuzzy k-means, Canopy, Dirichlet, and Mean-Shift.
- Supports Distributed Naive Bayes and Complementary Naive Bayes classification implementations.
- Comes with distributed fitness function capabilities for evolutionary programming.
- Includes matrix and vector libraries.

3.2.1 Mahout's Story [27]

First, some background on Mahout itself is in order. You may be wondering how to pronounce Mahout: in the way it's commonly Anglicized, it should rhyme with trout. It's a Hindi word that refers to an elephant driver, and to explain that one, here's a little history.
Mahout began life in 2008 as a subproject of Apache's Lucene project, which provides the well-known open source search engine of the same name. Lucene provides advanced implementations of search, text mining, and information-retrieval techniques. In the universe of computer science, these concepts are adjacent to machine learning techniques like clustering and, to an extent, classification. As a result, some of the work of the Lucene committers that fell more into these machine learning areas was spun off into its

own subproject. Soon after, Mahout absorbed the Taste open source collaborative filtering project.
Figure 3.26 shows some of Mahout's lineage within the Apache Software Foundation. As of April 2010, Mahout became a top-level Apache project in its own right, and got a brand-new elephant rider logo to boot.
Much of Mahout's work has been not only implementing these algorithms conventionally, in an efficient and scalable way, but also converting some of these algorithms to work at scale on top of Hadoop. Hadoop's mascot is an elephant, which at last explains the project name!
Mahout incubates a number of techniques and algorithms, many still in development or in an experimental phase (https://cwiki.apache.org/confluence/display/MAHOUT/Algorithms). At this early stage in the project's life, three core themes are evident: recommender engines (collaborative filtering), clustering, and classification.
This is by no means all that exists within Mahout, but they are the most prominent and mature themes at the time of writing.

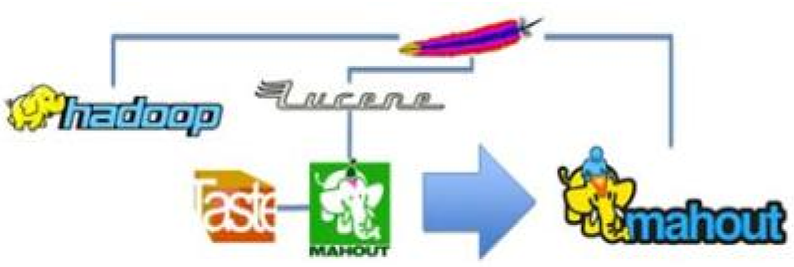


Figure 3.27 Apache Mahout and its related projects within the Apache Software Foundation. [27]

3.2.2 What is Machine Learning? [30]

Machine learning is a branch of science that deals with programming the systems in such a way that they automatically learn and improve with experience. Here, learning means recognizing and understanding the input data and making wise decisions based on the supplied data.

It is very difficult to cater to all the decisions based on all possible inputs. To tackle this problem, algorithms are developed. These algorithms build knowledge from specific data and past experience with the principles of statistics, probability theory, logic, combinatorial optimization, search, reinforcement learning, and control theory.

The developed algorithms form the basis of various applications such as:

- Vision processing
- Language processing
- Forecasting (e.g., stock market trends)
- Pattern recognition
- Games
- Data mining
- Expert systems
- Robotics

Machine learning is a vast area and it is quite beyond the scope of this tutorial to cover all its features. There are several ways to implement machine learning techniques, however the most commonly used ones are supervised and unsupervised learning.

Supervised Learning [30]

Supervised learning deals with learning a function from available training data. A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. Common examples of supervised learning include:

- classifying e-mails as spam,
- labeling webpages based on their content, and
- voice recognition.

There are many supervised learning algorithms such as neural networks, Support Vector Machines (SVMs), and Naive Bayes classifiers. Mahout implements Naive Bayes classifier.

Unsupervised Learning [30]

Unsupervised learning makes sense of unlabeled data without having any predefined dataset for its training. Unsupervised learning is an extremely powerful tool for analyzing available data and look for patterns and trends. It is most commonly used for clustering similar input into logical groups. Common approaches to unsupervised learning include:

- k-means
- self-organizing maps, and
- hierarchical clustering

Recommendation [30]

Recommendation is a popular technique that provides close recommendations based on user information such as previous purchases, clicks, and ratings.

Amazon uses this technique to display a list of recommended items that you might be interested in, drawing information from your past actions. There are recommender engines that work behind Amazon to capture user behavior and recommend selected items based on your earlier actions.

Facebook uses the recommender technique to identify and recommend the "people you may know list".

Classification [30]

Classification, also known as categorization, is a machine learning technique that uses known data to determine how the new data should be classified into a set of existing categories. Classification is a form of supervised learning.

Mail service providers such as Yahoo! and Gmail use this technique to decide whether a new mail should be classified as a spam. The categorization algorithm trains itself by analyzing user habits of marking certain mails as spams. Based on that, the classifier decides whether a future mail should be deposited in your inbox or in the spams folder.

iTunes application uses classification to prepare playlists.

Clustering [30]

Clustering is used to form groups or clusters of similar data based on common characteristics. Clustering is a form of unsupervised learning.

Search engines such as Google and Yahoo! use clustering techniques to group data with similar characteristics.

Newsgroups use clustering techniques to group various articles based on related topics.

The clustering engine goes through the input data completely and based on the characteristics of the data, it will decide under which cluster it should be grouped. Take a look at the following example.

3.2.3 Mahout's Machine Learning Themes [27, 30]

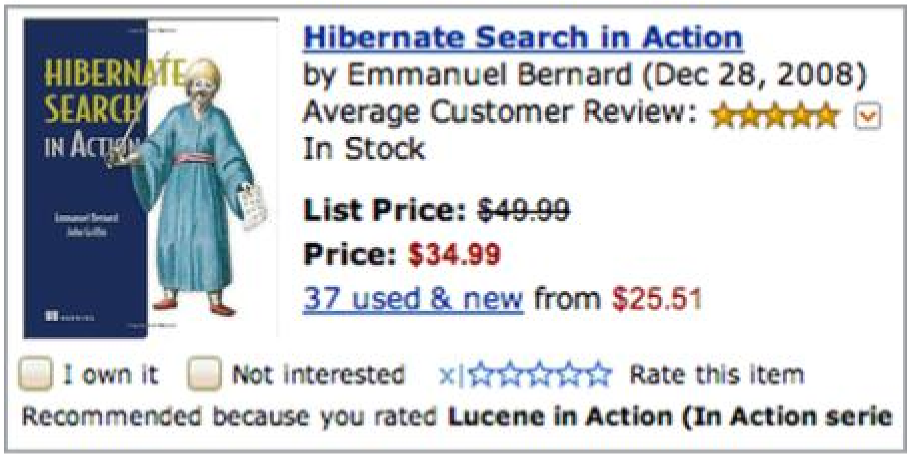
Although Mahout is, in theory, a project open to implementations of all kinds of machine learning techniques, it's in practice a project that focuses on three key areas of machine learning at the moment. These are recommender engines (collaborative filtering), clustering, and classification.

3.2.3.1 Recommender engines [27]

Recommender engines are the most immediately recognizable machine learning technique in use today. You'll have seen services or sites that attempt to recommend books or movies or articles based on your past actions. They try to infer tastes and preferences and identify unknown items that are of interest:

- Amazon.com is perhaps the most famous e-commerce site to deploy recommendations. Based on purchases and site activity, Amazon recommends books and other items likely to be of interest. See figure 3.28.
- Netflix similarly recommends DVDs that may be of interest, and famously offered a \$1,000,000 prize to researchers who could improve the quality of their recommendations.
- Dating sites like Líbímseti can even recommend people to people.
- Social networking sites like Facebook use variants on recommender techniques to identify people most likely to be as-yet-unconnected friends.

As Amazon and others have demonstrated, recommenders can have concrete commercial value by enabling smart cross-selling opportunities. One firm reports that recommending products to users can drive an 8 to 12 percent increase in sales.[28]



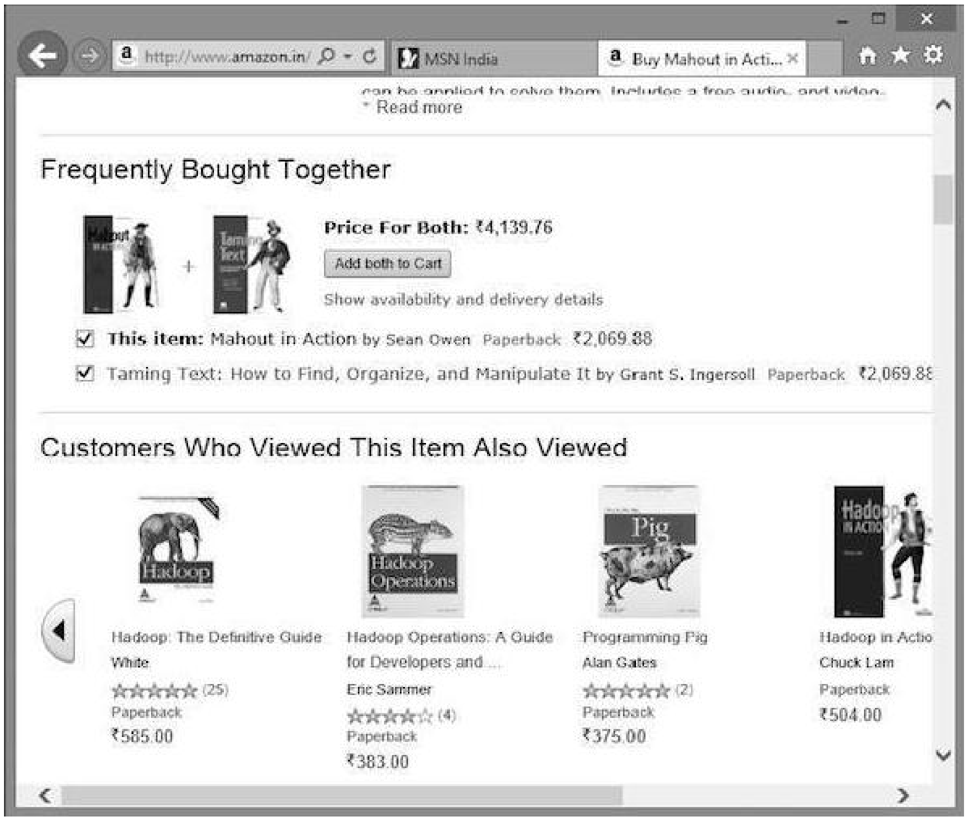
Hibernate Search in Action
by Emmanuel Bernard (Dec 28, 2008)
Average Customer Review: ★★★★★
In Stock

List Price: ~~$49.99~~
Price: $34.99
37 used & new from $25.51

I own it Not interested x|☆☆☆☆☆ Rate this item
Recommended because you rated Lucene in Action (In Action serie

Figure 3.28 A recommendation from Amazon. Based on past purchase history and other activity of customers like the user, Amazon considers this to be something the user is interested in. It can even list similar items that the user has bought or liked that in part caused the recommendation. [27]

Suppose you want to purchase the book "Mahout in Action" from Amazon. Along with the selected product, Amazon also displays a list of related recommended items, as shown below.



http://www.amazon.in/ MSN India Buy Mahout in Acti...

Read more

Frequently Bought Together

Price For Both: ₹4,139.76
Add both to Cart
Show availability and delivery details

☑ This item: Mahout in Action by Sean Owen Paperback ₹2,069.88
☑ Taming Text: How to Find, Organize, and Manipulate It by Grant S. Ingersoll Paperback ₹2,069.88

Customers Who Viewed This Item Also Viewed

Hadoop: The Definitive Guide
White
★★★★★ (25)
Paperback
₹585.00

Hadoop Operations: A Guide for Developers and ...
Eric Sammer
★★★★☆ (4)
Paperback
₹383.00

Programming Pig
Alan Gates
★★★★★ (2)
Paperback
₹375.00

Hadoop in Actio
Chuck Lam
Paperback
₹504.00

Figure 3.29 [30]

Such recommendation lists are produced with the help of recommender engines. Mahout provides recommender engines of several types such as:

- user-based recommenders,
- item-based recommenders, and
- several other algorithms.
- Mahout Recommender Engine

Mahout has a non-distributed, non-Hadoop-based recommender engine. You should pass a text document having user preferences for items. And the output of this engine would be the estimated preferences of a particular user for other items.

Example [30]
Consider a website that sells consumer goods such as mobiles, gadgets, and their accessories. If we want to implement the features of Mahout in such a site, then we can build a recommender engine. This engine analyzes past purchase data of the users and recommends new products based on that.

The components provided by Mahout to build a recommender engine are as follows:

- DataModel
- UserSimilarity
- ItemSimilarity
- UserNeighborhood
- Recommender

From the data store, the data model is prepared and is passed as an input to the recommender engine. The Recommender engine generates the recommendations for a particular user. Given below is the architecture of recommender engine.

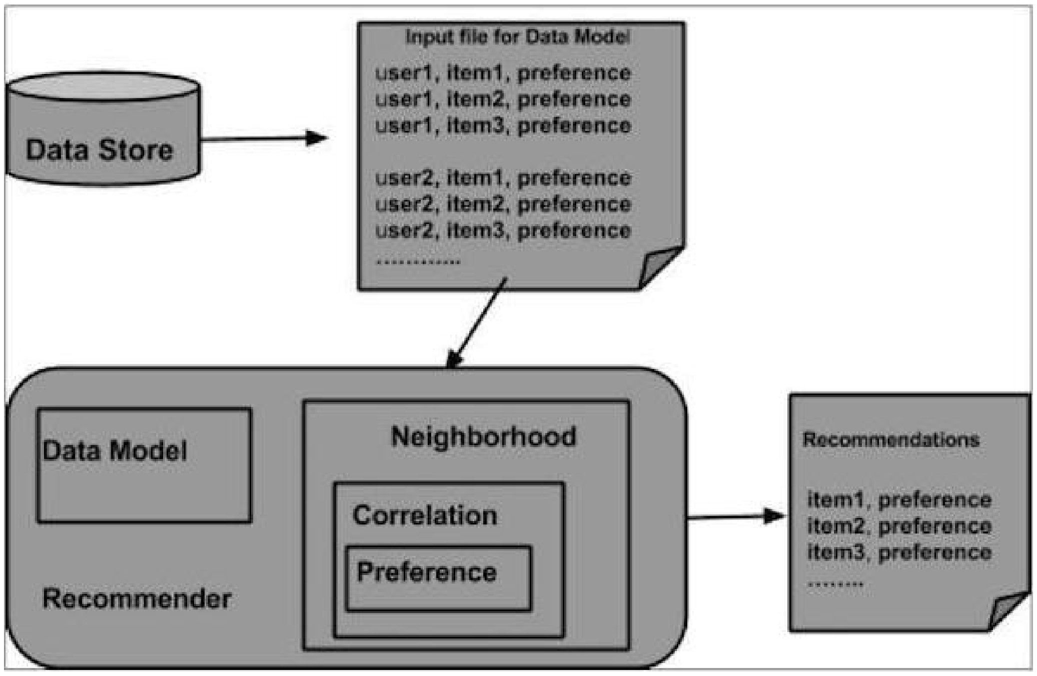


Figure 3.30 Architecture of Recommender Engine [30]

Building a Recommender using Mahout [30]

Here are the steps to develop a simple recommender:

Step1: Create DataModel Object

The constructor of **PearsonCorrelationSimilarity** class requires a data model object, which holds a file that contains the Users, Items, and Preferences details of a product. Here is the sample data model file:

```
1,00,1.0
1,01,2.0
1,02,5.0
1,03,5.0
1,04,5.0

2,00,1.0
2,01,2.0
2,05,5.0
2,06,4.5
2,02,5.0

3,01,2.5
3,02,5.0
3,03,4.0
3,04,3.0

4,00,5.0
4,01,5.0
4,02,5.0
4,03,0.0
```

The **DataModel** object requires the file object, which contains the path of the input file. Create the **DataModel** object as shown below.

```
DataModel datamodel = new FileDataModel(new File("input file"));
```

Step2: Create UserSimilarity Object

Create **UserSimilarity** object using PearsonCorrelationSimilarity class as shown below:

```
UserSimilarity similarity = new PearsonCorrelationSimilarity(datamodel);
```

Step3: Create UserNeighborhood object

This object computes a "neighborhood" of users like a given user. There are two types of neighborhoods:

- **NearestNUserNeighborhood** - This class computes a neighborhood consisting of the nearest n users to a given user. "Nearest" is defined by the given UserSimilarity.

- **ThresholdUserNeighborhood** - This class computes a neighborhood consisting of all the users whose similarity to the given user meets or exceeds a certain threshold. Similarity is defined by the given UserSimilarity.

Here we are using **ThresholdUserNeighborhood** and set the limit of preference to 3.0.

```
UserNeighborhood neighborhood = new ThresholdUserNeighborhood(3.0, similarity, model);
```

Step4: Create Recommender Object

Create **UserbasedRecomender** object. Pass all the above created objects to its constructor as shown below.

```
UserBasedRecommender recommender = new GenericUserBasedRecommender(model,
neighborhood, similarity);
```

Step5: Recommend Items to a User

Recommend products to a user using the recommend() method of **Recommender** interface. This method requires two parameters. The first represents the user id of the user to whom we need to send the recommendations, and the second represents the number of recommendations to be sent. Here is the usage of **recommender()** method:

```
List<RecommendedItem> recommendations = recommender.recommend(2, 3);


for (RecommendedItem recommendation : recommendations) {
   System.out.println(recommendation);
 }
```

Example Program [30]

Given below is an example program to set recommendation. Prepare the recommendations for the user with user id 2.

```
import java.io.File;

import java.util.List;


import org.apache.mahout.cf.taste.impl.model.file.FileDataModel;

import org.apache.mahout.cf.taste.impl.neighborhood.ThresholdUserNeighborhood;

import org.apache.mahout.cf.taste.impl.recommender.GenericUserBasedRecommender;

import org.apache.mahout.cf.taste.impl.similarity.PearsonCorrelationSimilarity;


import org.apache.mahout.cf.taste.model.DataModel;

import org.apache.mahout.cf.taste.neighborhood.UserNeighborhood;


import org.apache.mahout.cf.taste.recommender.RecommendedItem;

import org.apache.mahout.cf.taste.recommender.UserBasedRecommender;


import org.apache.mahout.cf.taste.similarity.UserSimilarity;


public class Recommender {

  public static void main(String args[]){

    try{

      //Creating data model

      DataModel datamodel = new FileDataModel(new File("data")); //data


      //Creating UserSimilarity object.

      UserSimilarity usersimilarity = new PearsonCorrelationSimilarity(datamodel);
```

```
        //Creating UserNeighbourHHood object.

        UserNeighborhood userneighborhood = new ThresholdUserNeighborhood(3.0,
usersimilarity, datamodel);



        //Create UserRecomender

        UserBasedRecommender recommender = new
GenericUserBasedRecommender(datamodel, userneighborhood, usersimilarity);



        List<RecommendedItem> recommendations = recommender.recommend(2, 3);



        for (RecommendedItem recommendation : recommendations) {

          System.out.println(recommendation);

        }



     }catch(Exception e){}



  }

 }
```

Compile the program using the following commands:

```
javac Recommender.java
java Recommender
```

It should produce the following output:

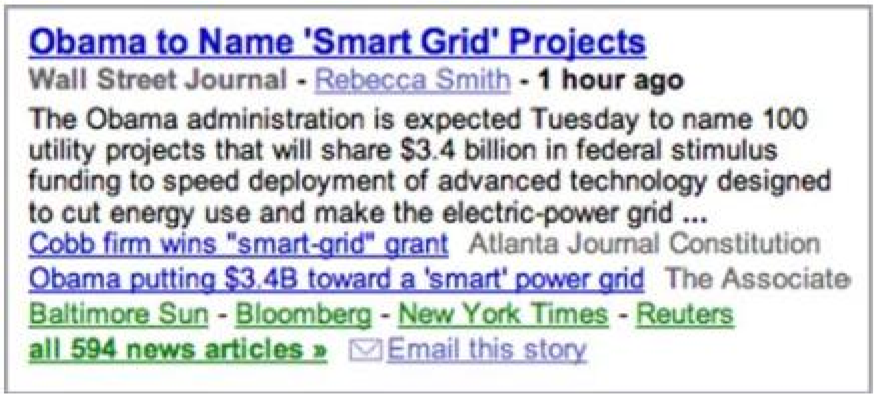
```
RecommendedItem [item:3, value:4.5]
RecommendedItem [item:4, value:4.0]
```

3.2.3.2 Clustering [30]

Clustering is less apparent, but it turns up in equally well-known contexts. As its name implies, clustering techniques attempt to group a large number of things together into clusters that share some similarity. It's a way to discover hierarchy and order in a large or hard-to-understand data set and in that way reveal interesting patterns or make the data set easier to comprehend.

- Google News groups news articles by topic using clustering techniques, in order to present news grouped by logical story, rather than presenting a raw listing of all articles. Figure 3.30 illustrates this.
- Search engines like Clusty group their search results for similar reasons.
- Consumers may be grouped into segments (clusters) using clustering techniques based on attributes like income, location, and buying habits.

Clustering helps identify structure, and even hierarchy, among a large collection of things that may be otherwise difficult to make sense of. Enterprises might use this technique to discover hidden groupings among users, or to organize a large collection of documents sensibly, or to discover common usage patterns for a site based on logs.



Obama to Name 'Smart Grid' Projects
Wall Street Journal - Rebecca Smith - 1 hour ago
The Obama administration is expected Tuesday to name 100 utility projects that will share $3.4 billion in federal stimulus funding to speed deployment of advanced technology designed to cut energy use and make the electric-power grid ...
Cobb firm wins "smart-grid" grant Atlanta Journal Constitution
Obama putting $3.4B toward a 'smart' power grid The Associate
Baltimore Sun - Bloomberg - New York Times - Reuters
all 594 news articles » Email this story

Figure 3.30 A sample news grouping from Google News. A detailed snippet from one representative story is displayed, and links to a few other similar stories within the cluster for this topic are shown. Links to all the stories that are clustered together in this topic are available too. [27]

Applications of Clustering

- Clustering is broadly used in many applications such as market research, pattern recognition, data analysis, and image processing.

- Clustering can help marketers discover distinct groups in their customer basis. And they can characterize their customer groups based on purchasing patterns.

- In the field of biology, it can be used to derive plant and animal taxonomies, categorize genes with similar functionality and gain insight into structures inherent in populations.

- Clustering helps in identification of areas of similar land use in an earth observation database.

- Clustering also helps in classifying documents on the web for information discovery.

- Clustering is used in outlier detection applications such as detection of credit card fraud.

- As a data mining function, Cluster Analysis serves as a tool to gain insight into the distribution of data to observe characteristics of each cluster.

Using Mahout, we can cluster a given set of data. The steps required are as follows:

- **Algorithm** You need to select a suitable clustering algorithm to group the elements of a cluster.

- **Similarity and Dissimilarity** You need to have a rule in place to verify the similarity between the newly encountered elements and the elements in the groups.

- **Stopping Condition** A stopping condition is required to define the point where no clustering is required.

Procedure of Clustering

To cluster the given data you need to -

- Start the Hadoop server. Create required directories for storing files in Hadoop File System. (Create directories for input file, sequence file, and clustered output in case of canopy).

- Copy the input file to the Hadoop File system from Unix file system.

- Prepare the sequence file from the input data.
- Run any of the available clustering algorithms.
- Get the clustered data.

Starting Hadoop

Mahout works with Hadoop, hence make sure that the Hadoop server is up and running.

```
$ cd HADOOP_HOME/bin
$ start-all.sh
```

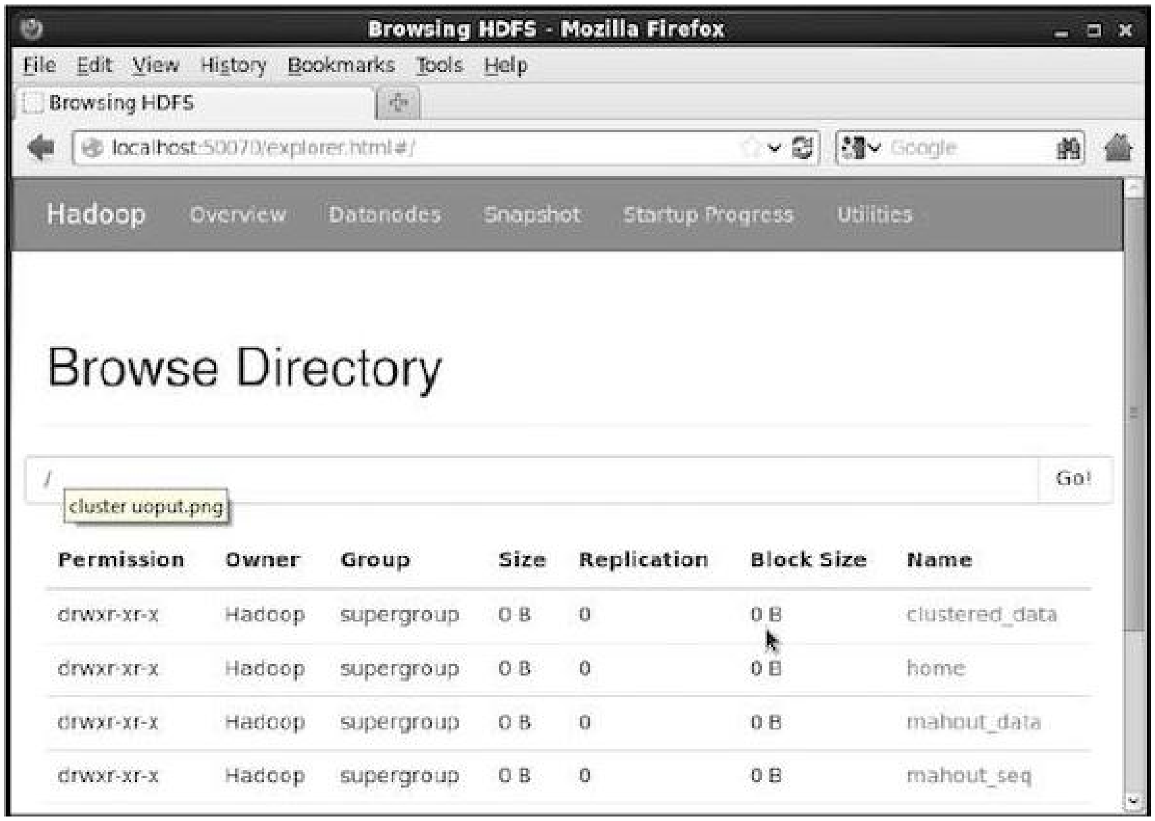
Preparing Input File Directories

Create directories in the Hadoop file system to store the input file, sequence files, and clustered data using the following command:

```
$ hadoop fs -p mkdir /mahout_data
$ hadoop fs -p mkdir /clustered_data
$ hadoop fs -p mkdir /mahout_seq
```

You can verify whether the directory is created using the hadoop web interface in the following URL - http://localhost:50070/

It gives you the output as shown below:


Browsing HDFS - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Browsing HDFS
localhost:50070/explorer.html#/
Google
Hadoop
Overview
Datanodes
Snapshot
Startup Progress
Utilities
Go!
cluster uoput.png

Browse Directory

/

Permission	Owner	Group	Size	Replication	Block Size	Name
drwxr-xr-x	Hadoop	supergroup	0 B	0	0 B	clustered_data
drwxr-xr-x	Hadoop	supergroup	0 B	0	0 B	home
drwxr-xr-x	Hadoop	supergroup	0 B	0	0 B	mahout_data
drwxr-xr-x	Hadoop	supergroup	0 B	0	0 B	mahout_seq

Copying Input File to HDFS

Now, copy the input data file from the Linux file system to mahout_data directory in the Hadoop File System as shown below. Assume your input file is mydata.txt and it is in the /home/Hadoop/data/ directory.

```
$ hadoop fs -put /home/Hadoop/data/mydata.txt /mahout_data/
```

Preparing the Sequence File

Mahout provides you a utility to convert the given input file in to a sequence file format. This utility requires two parameters.

- The input file directory where the original data resides.
- The output file directory where the clustered data is to be stored.

Given below is the help prompt of mahout **seqdirectory** utility.

Step 1: Browse to the Mahout home directory. You can get help of the utility as shown below:

```
[Hadoop@localhost bin]$ ./mahout seqdirectory --help
Job-Specific Options:
--input (-i) input Path to job input directory.
--output (-o) output The directory pathname for output.
--overwrite (-ow) If present, overwrite the output directory
```

Generate the sequence file using the utility using the following syntax:

```
mahout seqdirectory -i <input file path> -o <output directory>
```

Example

```
mahout seqdirectory
-i hdfs://localhost:9000/mahout_seq/
-o hdfs://localhost:9000/clustered_data/
```

Clustering Algorithms [30]

Mahout supports two main algorithms for clustering namely:

- Canopy clustering
- K-means clustering

Canopy Clustering [30]

Canopy clustering is a simple and fast technique used by Mahout for clustering purpose. The objects will be treated as points in a plain space. This technique is often used as an initial step in other clustering techniques such as k-means clustering. You can run a Canopy job using the following syntax:

```
mahout canopy -i <input vectors directory>
-o <output directory>
-t1 <threshold value 1>
-t2 <threshold value 2>
```

Canopy job requires an input file directory with the sequence file and an output directory where the clustered data is to be stored.

Example

```
mahout canopy -i hdfs://localhost:9000/mahout_seq/mydata.seq
-o hdfs://localhost:9000/clustered_data
-t1 20
-t2 30
```

You will get the clustered data generated in the given output directory.

K-means Clustering [30]

K-means clustering is an important clustering algorithm. The k in k-means clustering algorithm represents the number of clusters the data is to be divided into. For example, the k value specified to this algorithm is selected as 3, the algorithm is going to divide the data into 3 clusters.

Each object will be represented as vector in space. Initially k points will be chosen by the algorithm randomly and treated as centers, every object closest to each center are clustered. There are several algorithms for the distance measure and the user should choose the required one.

Creating Vector Files

- Unlike Canopy algorithm, the k-means algorithm requires vector files as input, therefore you have to create vector files.

- To generate vector files from sequence file format, Mahout provides the **seq2parse** utility.

Given below are some of the options of **seq2parse** utility. Create vector files using these options.

```
$MAHOUT_HOME/bin/mahout seq2sparse
--analyzerName (-a) analyzerName  The class name of the analyzer
--chunkSize (-chunk) chunkSize    The chunkSize in MegaBytes.
--output (-o) output              The directory pathname for o/p
--input (-i) input                Path to job input directory.
```

After creating vectors, proceed with k-means algorithm. The syntax to run k-means job is as follows:

```
mahout kmeans -i <input vectors directory>
-c  <input clusters directory>
-o  <output working directory>
-dm <Distance Measure technique>
-x  <maximum number of iterations>
-k  <number of initial clusters>
```

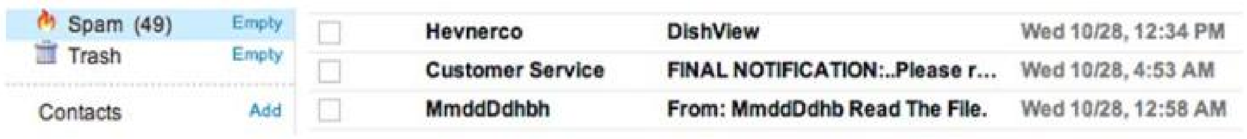
K-means clustering job requires input vector directory, output clusters directory, distance measure, maximum number of iterations to be carried out, and an integer value representing the number of clusters the input data is to be divided into.

3.2.3.3 Classification [30]

Classification techniques decide how much a thing is or isn't part of some type or category, or how much it does or doesn't have some attribute. Classification, like clustering, is ubiquitous, but it's even more behind the scenes. Often these systems learn by reviewing many instances of items in the categories in order to deduce classification rules. This general idea has many applications:

- Yahoo! Mail decides whether or not incoming messages are spam based on prior emails and spam reports from users, as well as on characteristics of the email itself. A few messages classified as spam are shown in figure 3.31.
- Google's Picasa and other photo-management applications can decide when a region of an image contains a human face.
- Optical character recognition software classifies small regions of scanned text into individual characters.
- Apple's Genius feature in iTunes reportedly uses classification to classify songs into potential playlists for users.

Classification helps decide whether a new input or thing matches a previously observed pattern or not, and it's often used to classify behavior or patterns as unusual.
It could be used to detect suspicious network activity or fraud. It might be used to figure out when a user's message indicates frustration or satisfaction.
Each of these techniques works best when provided with a large amount of good input data. In some cases, these techniques must not only work on large amounts of input, but must produce results quickly, and these factors make scalability a major issue.
And, as mentioned before, one of Mahout's key reasons for being is to produce implementations of these techniques that do scale up to huge input.



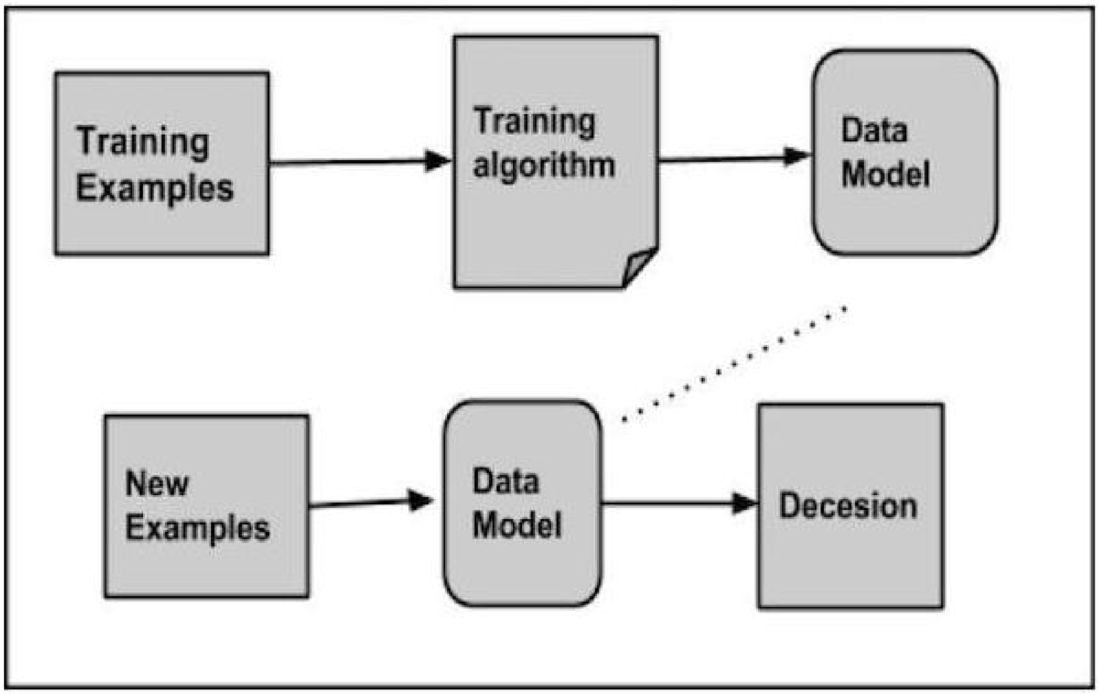
Hevnerco	DishView	Wed 10/28, 12:34 PM
Customer Service	FINAL NOTIFICATION:..Please r...	Wed 10/28, 4:53 AM
MmddDdhbh	From: MmddDdhb Read The File.	Wed 10/28, 12:58 AM

Figure 3.31 Spam messages as detected by Yahoo! Mail. Based on reports of email spam from users, plus other analysis, the system has learned certain attributes that usually identify spam. For example, messages mentioning "Viagra" are frequently spam—as are those with clever misspellings like "v1agra." The presence of such terms is an example of an attribute that a spam classifier can learn. [27]

How Classification Works [30]

While classifying a given set of data, the classifier system performs the following actions:

- Initially a new data model is prepared using any of the learning algorithms.
- Then the prepared data model is tested.
- Thereafter, this data model is used to evaluate the new data and to determine its class.



Applications of Classification [30]

- **Credit card fraud detection** - The Classification mechanism is used to predict credit card frauds. Using historical information of previous frauds, the classifier can predict which future transactions may turn into frauds.

- **Spam e-mails** - Depending on the characteristics of previous spam mails, the classifier determines whether a newly encountered e-mail should be sent to the spam folder.

Naive Bayes Classifier [30]

Mahout uses the Naive Bayes classifier algorithm. It uses two implementations:

- Distributed Naive Bayes classification
- Complementary Naive Bayes classification

Naive Bayes is a simple technique for constructing classifiers. It is not a single algorithm for training such classifiers, but a family of algorithms. A Bayes classifier constructs models to classify problem instances. These classifications are made using the available data.

An advantage of naive Bayes is that it only requires a small amount of training data to estimate the parameters necessary for classification.

For some types of probability models, naive Bayes classifiers can be trained very efficiently in a supervised learning setting.

Despite its oversimplified assumptions, naive Bayes classifiers have worked quite well in many complex real-world situations.

Procedure of Classification [30]

The following steps are to be followed to implement Classification:

- Generate example data
- Create sequence files from data
- Convert sequence files to vectors
- Train the vectors
- Test the vectors

Step1: Generate Example Data
Generate or download the data to be classified. For example, you can get the 20 newsgroups example data from the following link: http://people.csail.mit.edu/jrennie/20Newsgroups/20news-bydate.tar.gz

Create a directory for storing input data. Download the example as shown below.

```
$ mkdir classification_example
$ cd classification_example
$tar xzvf 20news-bydate.tar.gz
wget http://people.csail.mit.edu/jrennie/20Newsgroups/20news-bydate.tar.gz
```

Step 2: Create Sequence Files
Create sequence file from the example using **seqdirectory** utility. The syntax to generate sequence is given below:

```
mahout seqdirectory -i <input file path> -o <output directory>
```

Step 3: Convert Sequence Files to Vectors
Create vector files from sequence files using **seq2parse** utility. The options of **seq2parse** utility are given below:

```
$MAHOUT_HOME/bin/mahout seq2sparse
--analyzerName (-a) analyzerName  The class name of the analyzer
--chunkSize (-chunk) chunkSize    The chunkSize in MegaBytes.
--output (-o) output              The directory pathname for o/p
--input (-i) input                Path to job input directory.
```

Step 4: Train the Vectors
Train the generated vectors using the **trainnb** utility. The options to use **trainnb** utility are given below:

```
mahout trainnb
 -i ${PATH_TO_TFIDF_VECTORS}
 -el
 -o ${PATH_TO_MODEL}/model
 -li ${PATH_TO_MODEL}/labelindex
 -ow
 -c
```

Step 5: Test the Vectors
Test the vectors using **testnb** utility. The options to use **testnb** utility are given below:

```
mahout testnb
 -i ${PATH_TO_TFIDF_TEST_VECTORS}
 -m ${PATH_TO_MODEL}/model
 -l ${PATH_TO_MODEL}/labelindex
 -ow
 -o ${PATH_TO_OUTPUT}
 -c
 -seq
```

3.2.4 Tackling large scale with Mahout and Hadoop [27]

How real is the problem of scale in machine learning algorithms? Let's consider the size of a few problems where you might deploy Mahout.

Consider that Picasa may have hosted over half a billion photos even three years ago, according to some crude estimates.[29] This implies millions of new photos per day that must be analyzed. The analysis of one photo by itself isn't a large problem, even though it's repeated millions of times. But the learning phase can require information from each of the billions of photos simultaneously—a computation on a scale that isn't feasible for a single machine.

According to a similar analysis, Google News sees about 3.5 million new news articles per day. Although this does not seem like a large amount in absolute terms, consider that these articles must be clustered, along with other recent articles, in minutes in order to become available in a timely manner.

The subset of rating data that Netflix published for the Netflix Prize contained 100 million ratings. Because this was just the data released for contest purposes, presumably the total amount of data that Netflix actually has and must process to create recommendations is many times larger!

Machine learning techniques must be deployed in contexts like these, where the amount of input is large—so large that it isn't feasible to process it all on one computer, even a powerful one. Without an implementation such as Mahout, these would be impossible tasks. This is why Mahout makes scalability a top priority. Sophisticated machine learning techniques, applied at scale, were until recently only something that large, advanced technology companies could consider using. But today computing power is cheaper than ever and more accessible via open source frameworks like Apache's Hadoop. Mahout attempts to complete the puzzle by providing quality, open source implementations capable of solving problems at this scale with Hadoop, and putting this into the hands of all technology organizations.

Some of Mahout makes use of Hadoop, which includes an open source, Java-based implementation of the MapReduce distributed computing framework popularized and used internally at Google.

MapReduce is a programming paradigm that at first sounds odd, or too simple to be powerful. The MapReduce paradigm applies to problems where the input is a set of key-value pairs. A map function turns these key-value pairs into other intermediate key value pairs. A reduce function merges in some way all values for each intermediate key to produce output. Actually, many problems can be framed as MapReduce problems, or as a series of them. The paradigm also lends itself quite well to parallelization: all of the processing is independent and so can be split across many machines.

Hadoop implements the MapReduce paradigm, which is no small feat, even given how simple MapReduce sounds. It manages storage of the input, intermediate key value pairs, and output; this data could potentially be massive and must be available to many worker machines, not just stored locally on one. It also manages partitioning and data transfer between worker machines, as well as detection of and recovery from individual machine failures. Understanding how much work goes on behind the scenes will help prepare you for how relatively complex using Hadoop can seem. It's not just a library you add to your project. It's several components, each with libraries and (several) standalone server processes, which might be run on several machines.

Operating processes based on Hadoop isn't simple, but investing in a scalable, distributed implementation can pay dividends later: your data may quickly grow to great size, and this sort of scalable implementation is a way to future-proof your application.