# Distributed Database Management Systems: The CAP Theorem

## Distributed Database Management Systems (DBMS)

**Parallel DBMS:** Database management systems that r8ely on the use of multiple processing cores and shared main memory and second storage to speed up data processing tasks.

**Distributed DBMS:** Database management systems that operate over a collection of loosely coupled nodes that share no physical components.

**Shared nothing architecture.** The distributed DBMS model we consider is shared nothing. Its properties are:

- Multiple compute nodes
- Each node has its own private main memory (RAM) buffer dedicated data management tasks
- Each node has its own private secondary storage (disk storage) dedicated to data storage
- Nodes cannot directly access main memory or secondary storage that belongs to other nodes
- Nodes communicate to each other via messages
- The network is faulty: not all messages can arrive

**Building Distributed DBMS.** When building distributed DBMS, the following challenges need to be addressed:

1. **Data storage.** How is data distributed among the private disk partitions of individual nodes? How is data accessed?

2. **Transaction management.** DBMS operate in terms of atomic transactions: sets of operations that must either be all completed together, or not performed at all if at least one operation fails. DBMS usually manage multiple transactions at a time via a *concurrency control* mechanism (covered in detail in CSC 468). When two or more transactions want to work with *the same data*, conflicts may arise. The DBMS implement transaction scheduling, concurrency control, and conflict resolution procedures that satisfy the so called ACID properties:

   - **A** for **Atomicity**: either all operations in a transaction must be performed, or none.
   - **C** for **Consistency**: all data written to the database must be a result of a valid operation (otherwise specified as: "a transaction that starts in a valid database state must end in a valid database state")
   - **I** for **Isolation**: the outcomes of one transaction must not depend on the outcomes of any other transaction (transactions run in isolation, or "concurrent execution of transactions leave the database in the same state the would have been obtained if the transactions were executed sequentially").
   - **D** for **Durability**: one the transaction has been committed, its effects on the database state must persist even through a system failure.

   Non-distributed DBMS implement a lot of controls to support ACID properties, but at the cost of efficiency. This cost rises significantly in the presence of distributed data. Should ACID properties be retained in distributed DBMS, and if yes - how? If not, which properties to keep and how to gracefully abandon the remaining ones?

3. **Fault-tolerance.** How can a distributed DBMS deal with loss of messages? How can it deal with loss of individual nodes?

4. **Query processing.** How is query execution distributed among the nodes?

## Consistency, Availability, Partition-Tolerance

In 2000[1], Eric Brewer[1] observed, that distributed systems[2] have three desired properties: consistency, availability, partition tolerance.

**Consistency:** is informally defined[3] as

> *all nodes see the same data all the time.*

Another way to define consistency is by saying that *the DBMS returns the correct response to each information request*, where *correct* is defined w.r.t. the semantics of the data.

[1] Actually, even earlier than that.

[2] Any distributed systems, not just distributed DBMS

[3] This is a different definition of "consistency" than the one used for "C" in "ACID". There is a long story here.

**Availability:** is defined as

> *each request eventually receives a response.*

While the presence of the word *eventually* appears to suggest that this is not a very strong condition, it is nevertheless strong enough to cause trouble in the presence of faulty networks.

**Partition tolerance** is defined as

> *the system is able to operate despite arbitrary loss of connectivity between its parts.*

This appears to be a "strong" condition, because it asks the system to tolerate faults of arbitrary scope.

## The CAP Theorem

In 2000[4] Brewer presented the CAP Theorem formulated as follows[1]:

**Theorem.** In any shared data[5] system you can have **at most two** of the following properties: Consistency, Availability, Partition tolerance.

Later reviews of the meaning of the theorem made the following observation:

> A distributed system **must be** tolerant to partitions.

Based on this observation, one may restate The CAP theorem as follows:

**Theorem.** In any shared data system, in order to ensure Partition Tolerance, either Availability or Consistency must be sacrificed.

Gilbert and Lynch[2] proved a version of the CAP theorem, in which the notions of shared data system, availability, partition tolerance and consistency have been formally defined (on one hand), but (on the other hand), the proof appears to apply only to a subset of situations that can happen "in real life".

## Two Out of Three Ain't Bad?

Brewer[1] outlined the properties of the *three* categories of systems:

**Consistent Available (CA) systems:** essentially *non-distributed* single-node DBMS.

**Consistent Partition-tolerant (CP) systems:** distributed relational databases that try to enforce ACID properties on the system. Network communication protocols.

[4] Again, there is evidence that the actual conjecture was formulated as early as 1998.
[5] Brewer's euphemism for a distributed system in this context.

**Available Partition-tolerant (AP) systems:** systems that do not need a consistency guarantee to return a "good" response. World Wide Web. DNS.

As the new reading of the CAP theorem suggests, **when building distributed DBMS** we have a choice between CP and AP systems.

## References


[1] Eric A. Brewer, (2000), *Towards Robust Distributed Systems*, Invited Talk.

[2] S. Gilbert, N. Lynch (2002) Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services, *SIGACT News* 33(2): 51-59