

VICTORIA UNIVERSITY OF WELLINGTON  
*Te Whare Wananga o te Upoko o te Ika a Maui*



# ***Trade-offs in Cloud Databases***

***Lecturer : Dr. Pavle Mogin***

SWEN 432  
*Advanced Database Design and  
Implementation*

# ***Plan for Trade-offs in Cloud Databases***

---

- Features of relational SQL databases
  - ACID properties
  - NoSQL databases
  - CAP Theorem
  - BASE Properties
    - More about consistency/availability trade-offs
- 
- ***Readings:*** Have a look at *Useful Links at the Course Home Page*

# ***Traditional Databases***

---

- Most traditional databases are relational
- Relational data model has been developed to serve applications that insist on data consistency and reliability
- RDBMSs are OLTP DBMSs
- Data consistency and reliability are guaranteed by ACID properties
- ACID stands for **A**tomicity, **C**onsistency, **I**solation, and **D**urability

# ACID Database Properties

---

- **Atomicity**
  - Transactions must act as a whole, or not at all
  - No changes within a transaction can persist if any change within the transaction fails
- **Consistency**
  - Changes made by a transaction respect all database integrity constraints and the database remains in a consistent state at the end of a transaction
- **Isolation**
  - Transactions cannot see changes made by other concurrent transactions ,
  - Concurrent transactions leave the database in the same state as if the transactions were executed serially
- **Durability**
  - Database data are persistent,
  - Changes made by a transaction that completed successfully are stored permanently

# Techniques to Achieve ACID Properties

---

- Relational databases:
  - Have a schema that provides a strict structure and integrity constraints for database data,
  - Schemes are normalized to avoid data redundancy and update anomalies,
  - Normalization is achieved by a vertical splitting of database tables,
  - Queries frequently ask for joins.
- Integrity constraints (checked by writes) provide for consistency
- Locking (to avoid transaction anomalies, e.g. lost update) provides for isolation
- Logging – provides for durability
  - But changes made by each transaction have to be written on disk **two** times

# ***ACID Properties and Cloud Databases***

---

- Many cloud applications (like social networks) require databases that:
  - Are highly scalable and available,
  - Have high throughput, and
  - Be network partition tolerant
- These requirements are in a direct collision with ACID properties, particularly with integrity constraint checking, locking, logging, and performing joins
- These applications also consider strict data structuring not being suitable
- Also, consistency is considered to be of some lesser importance

# NoSQL Database Systems

---

- Many cloud databases are not relational
- They are termed **NoSQL** databases (**Not only SQL**)
  - NoSQL does not mean “NO to SQL”, but NO to “one size fits all”,
  - Different applications need different database paradigms
- Generally, NoSQL database systems:
  - Adopt the key-value data model for storing semi structured or unstructured data
  - Are either scheme less, or their schemes lack rigidity and integrity constraints of relational schemes
  - Use data partitioning, replication and distribution to several independent network nodes made of commodity machines for:
    - Scalability,
    - Availability, and
    - Network partition tolerance
  - Are not normalized, and do not support joins but **store all data needed to answer a query in the same partition** to achieve a high throughput

# CAP Theorem

---

- NoSQL database systems do not support ACID properties
  - Locking and logging would severely impair scalability, availability, and throughput
- Still, **C**onsistency, **A**vailability, and Network **P**artition Tolerance (CAP) represent their desirable properties
- But, according the CAP Theorem, also known as Brewer's Conjecture, any networked, shared data system can have at most two of these three desirable properties



# CAP Theorem

(2)

- For a distributed database system as a service, the components of the CAP acronym stand for:
  - **Consistency** - A service is considered to be **consistent** if after an update operation of some writer all readers see the update in some shared data source (all nodes containing replicas of a datum have the same value),
  - **Availability** - A service is considered to have a high availability if it allows read and write operations a high proportion of time (even in the case of a node crash or some hardware or software parts are down due to upgrades)
  - **Network partition tolerance** is the ability of a service to perform expected operations in the presence of a network partition, unless the whole network crashes
    - A network partition occurs if two or more “islands” of network nodes cannot connect to each other
    - Dynamic addition and removal of nodes is also considered to be a network partition

# Comments on CAP Properties

---

- Contrary to traditional databases, many cloud database are not expected to satisfy any integrity constraints
- High availability of a service is often characterized by small latency
  - Amazon claims that raise of 0.1 sec in their response time will cost them 1% in sales,
  - Google claims that just .5 sec in latency caused traffic to drop by 20%
- It is hard to draw a clear border between availability and network partitions
  - Network partitions may induce delays in operations of a service, and hence a greater latency

# An Informal Proof of the CAP Theorem (1)

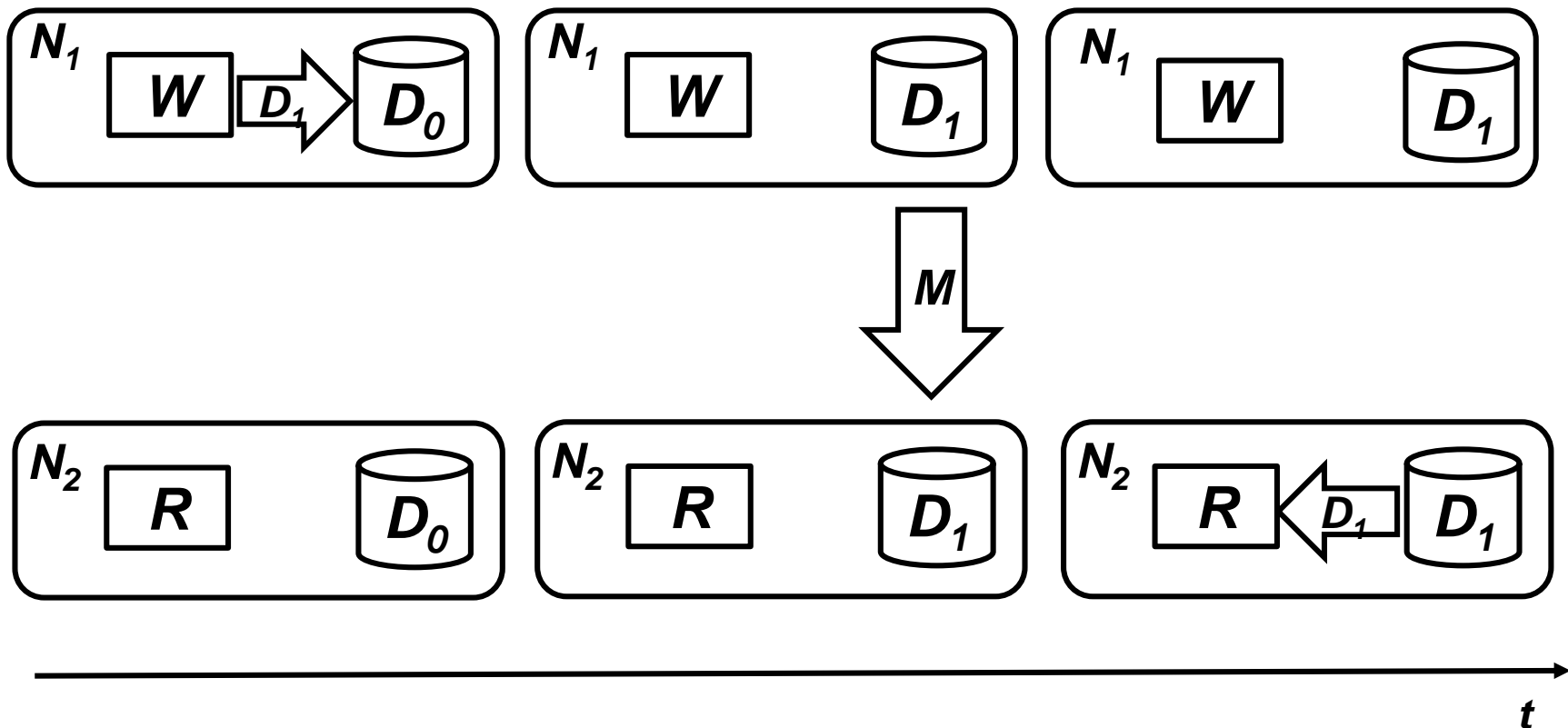
---

- Eric Brewer presented his conjecture in 2000, Gilbert&Lynch proved it formally in 2002
- Here is an informal presentation of their proof showing that a highly available and partition tolerant service may be inconsistent
- Let us consider two nodes  $N_1$  and  $N_2$ 
  - Both nodes contain the same copy of data  $D$  having a value  $D_0$ ,
  - A program  $W$  (write) runs on  $N_1$
  - A program  $R$  (read) runs on  $N_2$



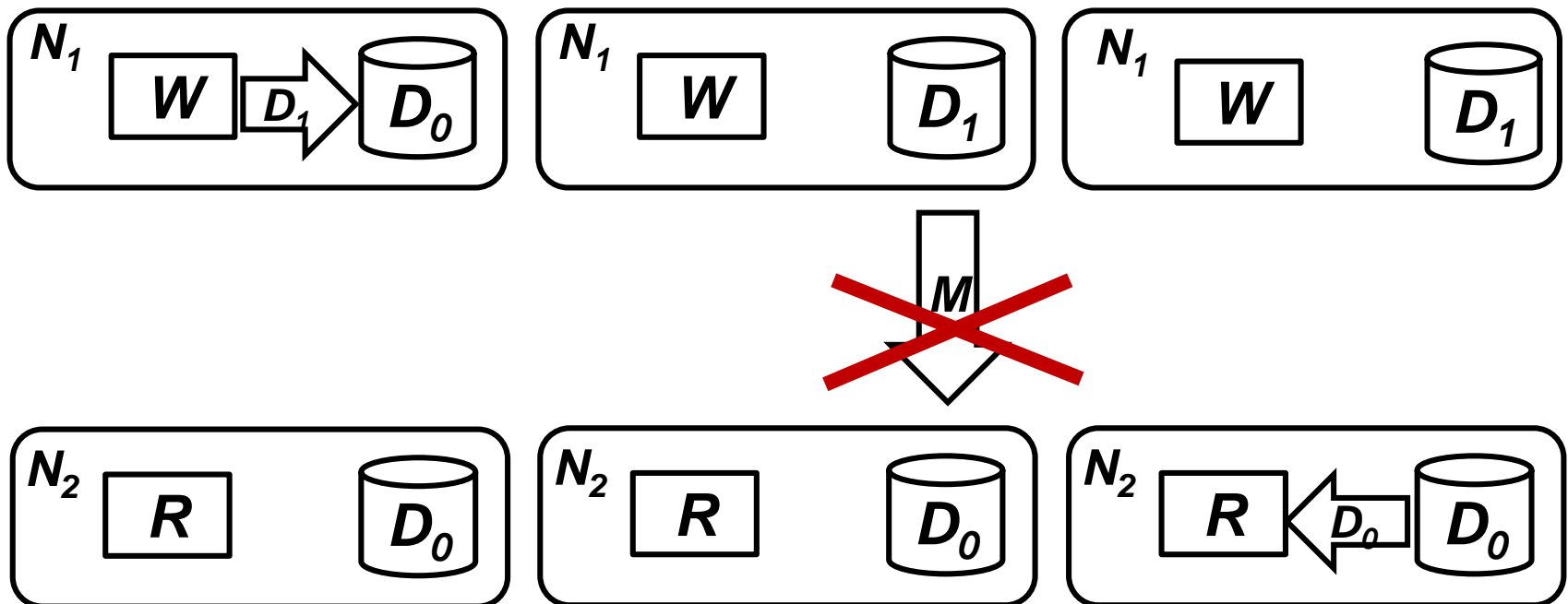
# An Informal Proof of the CAP Theorem (2)

- In a regular situation:
  - The program  $W$  writes a new value of  $D$ , say  $D_1$  on  $N_1$
  - Then,  $N_1$  sends a message  $M$  to  $N_2$  to update the copy of  $D$  to  $D_1$ ,
  - The program  $R$  reads  $D$  and returns  $D_1$



# An Informal Proof of the CAP Theorem (3)

- Assume now the network partitions and the message from  $N_1$  to  $N_2$  is not delivered



- The program  $R$  reads  $D$  and returns  $D_0$

# Comments on the Informal Proof (1)

---

- The service is **highly available**, since it performed both W and R without delay.
- The service is **partition tolerant** since it performed its tasks in spite the network partition.
- The service is **inconsistent**, since it delivered stale data.

# Comments on the CAP Theorem (2)

---

- If an application in a distributed and replicated network have to be:
  - Highly available (i.e. working with minimal latency), and
  - Tolerant to network partitions (performs expected user initiated operations), then
  - It has to perform an update even if not all nodes are able to acknowledge it
  - Hence the (strict) consistency can't be achieved
  - So, it may happen that some of  $N_i$  ( $i = 1, 2, \dots$ ) nodes have stale data, or even are unavailable
- The decision on which of {CA, CP, AP} property pairs to satisfy is made in early stages of a cloud application design, according to requirements and expectations defined for the application
- Most NoSQL database systems fall either in the AP class, or in the CP class

# **BASE Properties of Cloud DBs** (1)

---

- BASE (as defined by Eric Brewer) stands for:
  - **BA** (**B**asically **A**vailable) meaning that the system is always available, although there may be intervals of time when some of its components are not available
    - A basically available system answers each request in such a way that it appears that system works correctly and is always available and responsive
  - **S** (**S**oft State) means that the system has not to be in a consistent state after each transaction
    - The database state may change without user's intervention (due to an automatic replica convergence)
  - **E** (**E**ventually Consistent) guarantees that:
    - If no new updates are made to a given data item, accesses to each replica of that item will **eventually** return the last updated value
    - A system that has eventually achieved consistency is often said to have **converged**, or achieved **replica convergence**



# ***BASE Properties of Cloud DBs*** **(2)**

---

- The reconciliation of differences between multiple replicas requires exchanging and comparing versions of data between nodes
  - Usually "last writer wins"
  - The reconciliation is performed as a:
    - read,
    - write, or
    - asynchronous repair
    - Asynchronous repairs are performed in a background non transactional process
- ***Stale*** and ***approximate*** data in answers are tolerated

# Consistency – Availability Trade-offs (1)

---

- Assume:
  - A cloud database is divided into  $p$  partitions and each partition is replicated on  $n (\geq 3)$  servers (replica nodes),
  - The database is unavailable if any of its parts is unavailable,
  - Each node stores only one replica (to simplify analysis), and
  - Each reader may contact any of  $n$  replica nodes in a partition

# Consistency – Availability Trade-offs (2)

- **Strict** consistency: any read of a shared data item  $X$  returns the value stored by the most recent write on  $X$ 
  - Requires  $n$  servers to be available in each partition
  - This is hard to achieve with commodity hardware,
  - If only one node fails, the database is unavailable
- **Strong** consistency requires  $w + r > n$  for each partition
  - $w$  and  $r$  are numbers of nodes acknowledging a write and returning a read, respectively
  - If the condition  $w + r > n$  has been satisfied for each partition, at least 1 node in each partition can fail and the database is still available
  - A special case: **quorum** consistency, requires at least  $q = \lfloor n/2 \rfloor + 1$  replica nodes to be available in each partition
  - For  $w = n$  and  $r = 1$ , strong consistency turns into the strict one
- **Eventual** consistency requires at least 1 node to be available in each partition in every moment
  - At least one node has to acknowledge a write,
  - At least one node has to return a read,
  - These two nodes do not have to be the same

# Example: Consistency vs. Availability (1)

- Let the number of partitions  $p = 10$  and the replication factor  $n = 3$ 
  - So, the database is stored on 30 nodes
- Assume a client has requested the service to have **strong** consistency under **quorum**
- Question
  - How many nodes can fail and still the whole database to be available?
- Answer:
  - Since  $n = 3$ , quorum  $q = 2$ ,
  - In the **worst case**, all unavailable nodes belong to the same partition
    - Then, to have the whole database available under strong consistency, only **1** node in total is allowed to be unavailable for both writing and reading
  - In the **best case**, all available nodes are evenly distributed over all partitions
    - Then, to have the whole database available under strong consistency, one node in each partition is allowed to be unavailable for both writing and reading, allowing at most **10** nodes being simultaneously unavailable in total

## Example: Consistency vs. Availability (2)

---

- Let the number of partitions  $p = 10$  and the replication factor  $n = 3$ 
  - So, the database is stored on 30 nodes
- Assume a client has requested the service to has **eventual** consistency
- Question
  - How many nodes can fail and still the whole database to be available?
- Answer:
  - In the **worst case**, all unavailable nodes belong to the same partition
    - Then, to have the whole database available under eventual consistency, only **2** nodes in total are allowed to be unavailable
  - In the **best case**, all unavailable nodes are evenly distributed over all partitions
    - Then, to have the whole database available under eventual consistency, **2** nodes in each partition are allowed to be unavailable, giving **20** nodes in total

# Summary

(1)

- NoSQL cloud databases are scalable, highly available and tolerate network partitions
- Cloud database system can have only two out of **C**onsistency, **A**vailability, and Network **P**artition Tolerance
- BASE represents a “best effort” approach to ensuring database reliability
  - BASE forfeits ACID properties, also by its name (base versus acid, as found in chemistry)
  - **B**asically **A**vailable,
  - **S**oft state,
  - **E**ventually consistent

# Summary

(2)

- Eventual Consistency is a model used in distributed systems that guarantees that all accesses to replicas of an updated item will eventually return the last updated value
- Consistency levels:
  - Strict consistency,
  - Strong consistency,
  - Eventual consistency
- Consistency and availability mutually restrict each other:
  - As greater the consistency requirement as lower the database availability