Tradeoffs in Distributed Databases
Abstract

In a distributed database data is spread throughout the network into separated nodes with different DBMS systems (Date, 2000). According to CAP – theorem three database properties - consistency, availability and partition tolerance cannot be achieved simultaneously in distributed database systems. Two of these properties can be achieved but not all three at the same time (Brewer, 2000). Since this theorem there has been some development in network infrastructure. Also new methods to achieve consistency in distributed databases has emerged. This paper discusses trade-offs in distributed databases.

Keywords
Distributed database, cap theorem

Supervisor
Title, position First name Last name
# Contents

Abstract........................................................................................................................................... 2  
Contents ............................................................................................................................................... 3  
1. Introduction .................................................................................................................................. 4  
2. Distributed Databases .................................................................................................................. 5  
   2.1 Principles of distributed databases ......................................................................................... 6  
   2.2 Transaction management ....................................................................................................... 7  
      2.2.1 ACID transactions ........................................................................................................... 7  
      2.2.2 BASE transactions ......................................................................................................... 7  
      2.2.3 Paxos based non-blocking commit protocols ................................................................. 7  
      2.2.4 Isolation level ................................................................................................................ 8  
2.3 Replication .................................................................................................................................. 8  
2.4 Concurrency control .................................................................................................................. 9  
3. CAP Theorem .............................................................................................................................. 10  
   3.1 Overview of CAP theorem ....................................................................................................... 10  
      3.1.1 Weak consistency models .............................................................................................. 12  
   3.2 The Most Common sacrifice ..................................................................................................... 14  
      3.2.1 Partition tolerance ......................................................................................................... 14  
      3.2.2 Availability .................................................................................................................. 15  
      3.2.3 Consistency .................................................................................................................. 15  
3.3 Discussion ................................................................................................................................... 16  
4. Modern viewpoint ........................................................................................................................ 17  
   4.1 Proposals for distributed concurrency control and transaction handling ............................ 17  
   4.2 Effect on CAP theorem ............................................................................................................ 19  
5. Conclusions ..................................................................................................................................... 20  
References .......................................................................................................................................... 21  
Appendix A. Resources and links ..................................................................................................... 24
1. Introduction

In distributed database system data is spread across multiple sites connected via some sort of communication network system. In a distributed database data can be stored where it is mostly used yet it provides seamless access to data stored at remote sites. From user point of view using distributed database should be quite similar to using local or centralized database. Distribution increases database complexity (Date, 2000).

According to CAP – theorem it is not possible to achieve safety and liveness properties in an unreliable distributed system. You need to choose between consistency (safety) and availability (liveness) if there is network partitions (Gilbert & Lynch, 2012).

Traditional relational databases attempt to provide strong consistency via ACID transactions also in web environment. Operations are expected to either be executed completely or not at all (atomic), operations never result in inconsistent data, uncommitted transaction are isolated from each other and committed results are permanent (durable) (Gilbert & Lynch, 2002).

Todays web services are expected to be highly available. Users are not likely to visit again non responsive web pages (Lee & Lin, 2005). NoSql movement offers BASE transactions (basically available, soft state, eventually consistent) and reduced consistency models to ensure availability for web services (Pritchett, 2008).

In this paper distributed database and CAP – theorem are discussed. In particular which of the three properties, consistency, availability or partition tolerance is the most common property to be forfeited according to litterature. First distributed databases are treated. In the next chapter CAP- theorem is discussed in detail.

NoSql movement justify sacrificing consistency for better scalability. Also new proposals for distributed transactions and concurrency control in distributed environment has emerged. These proposals and their impact on CAP-theorem is studied in chapter 4.

Research questions:
“What is the most common property to be forfeited in CAP- theorem in litterature?”
“How new proposals for transaction management and concurrency control affect CAP-theorem?”

Literature study is used as research method in this paper. Material is searched using Google Scholar, Nelli portal, plain Google search and Research Gate.
2. Distributed Databases

A distributed database consists of independent sites which are connected with each other via some sort of communication network. Each site contains fully functional database that is capable of working by itself (Date, 2000). Figure 1 illustrates a typical distributed database system. In this chapter distributed database principles and basic concepts like transaction management and commit protocols are discussed. Also replication and lock and multiversion concurrency control methods will be explained.

![Figure 1 Distributed database system](image)

In distributed database data can be kept near where it is mostly used. This increases efficiency. Also accessibility increases because it is easy to access data stored in different location. On the other hand distribution of data increases complexity (Date, 2000.)
2.1 Principles of distributed databases

According to Date (2000) the fundamental principle of distributed database is that it should be used by the client just in the same way than the usage of ordinary centralized database. From this fundamental principle Date derives twelve subsidiary rules or objectives. These objectives are discussed next based on (Date, 2000) and (Foster & Godbole, 2014).

**Local autonomy**: All operations at a given site are controlled on that site only. This is not always possible, but sites should be autonomous to the maximum extent.

**No reliance on a central site**: Sites must be treated as equal. Central site becomes bottleneck and makes the system vulnerable for failure on the central site.

**Continuous operation**: The system should be able to run continuously. All site should never be down at the same time. This improves reliability and availability.

**Location independence**: User does not need to know where the data is located. The system should operate as if all data is stored locally on the user’s point of view.

**Fragmentation independence**: It should be possible to fragment data in relations to sites where it is most frequently used. This reduces network traffic. Fragmentation should be transparent to the user. In database literature, and also in this paper, term “partition” is often used instead of fragmentation.

**Replication independence**: It should be possible to replicate relations or fragments of relations to different sites to improve performance. Note that all replicas should be updated simultaneously to enforce consistency.

**Distributed query processing**: Distributed queries should be optimized between sites. Data should be transmitted a set at a time instead of record at a time. This lead to the fact that distributed database systems tend to be relational databases.

**Distributed transaction management**: In distributed system transaction can involve code execution and data updates on many sites. Major aspects of transaction management are recovery control and concurrency control. Transactions must be atomic, they must be fully executed (committed) or rolled back. Transaction management seems to play major role in CAP-theorem, so it shall be discussed in more detail in following sections.

**Hardware independence**: I should be possible to use different hardware platforms at different sites.

**Operating system independence**: It should be possible to run different operating system software at different sites.

**Network independence**: It should be possible to user different network hardware and protocols in different sites.

**DBMS independence**: Different DBMS suites may be used on different sites. These DBMS suites need to share common interface like ODBC or JDBC.
2.2 Transaction management

Transaction management is the major issue in distributed databases on CAP point of view. In this chapter ACID and BASE transaction and Paxos consensus protocol is discussed. Acid transactions and two phase commit makes things easy for application developers (Pritchett, 2008). Unfortunately 2PC is not well suitable for distributed database. New proposals for distributed transactions using Paxos protocol is discussed in chapter 4.

2.2.1 ACID transactions

ACID transactions guaranties consistency after every transaction and are easy for system developers to implement (Pritchett, 2008). According to Date (2002) relational databases provides the following ACID transaction properties. ACID model provides clear semantic for application developers to handle transactions.

Atomic: Transaction must be completed with successful commit on every site or rejected (rolled back) on every site.

Consistent: Database is consistent when it all data fills integrity rules (Brewer, 2012).

Isolation: Updates made by a transaction are not visible to other transactions until the transaction is committed. Isolation level can be adjusted in DBMS engine. More about isolation levels in chapter 2.3.6.

Durable: After successful commit new values are permanent and will survive event immediate system crash.

2.2.2 BASE transactions

BASE transactions provides availability for scalable applications (Pritchett, 2008). BASE transactions were originally designed by Eric Brewer and his coworkers (Brewer, 2012).

Basically Available: The system is always available to clients.

Soft state: Database nodes need not to be consistent all the time.

Eventually consistent: Eventually database nodes will be consistent. Amazon CTO Werner Vogel defines eventual consistency as “eventually all accesses return the last updated value” (Bailis & Ghodsi, 2013).

2.2.3 Paxos based non-blocking commit protocols.

Paxos consensus algorithm was introduced by Leslie Lamport in 1998 (Lamport, 1998). Paxos commit protocol is two phase commit protocol with multiple coordinators. In failure situation Paxos protocol can progress if majority of coordinators are operational. Paxos can progress if 2F+1 coordinators are available and no more than F coordinators fails. If more than F coordinators fails, then Paxos hangs until F coordinators are operational again. Special case of Paxos commit protocol with only one coordinator is essentially the same as two phase commit protocol (Gray & Lamport, 2005.) Figure 4 illustrates successful Paxos commit message sequence. Using Paxos protocol adds latency due to increased amount of
messages. Many modern distributed transaction handling systems use Paxos based protocols.

Figure 2 Paxos commit protocol

2.2.4 Isolation level

Isolation level defines how transactions are isolated from each other. In DBMS system these isolation levels are implemented by means of locking schemes (Date, 2000).

**Read uncommitted**: Query transaction uses no locks, it can see uncommitted, “dirty” data modified by other transaction. If the other transaction is rolled back the data may be lost.

**Read committed**: Query transaction can see only committed data. However, lock is released immediately after select is completed, so if transaction reads the same data again, it can have another value. This is called non **repeatable read** phenomenon.

**Repeatable read**: Query transaction can see only committed data, and read locks are held until the end of the transaction, so repeated reads for same data in that transaction result same value. Range locks are not used, so running the query again may end up with additional rows: **phantom data** phenomenon.

**Serializable**: Other transactions cannot have any impact on data accessed by a serializable transaction. In lock based concurrency management scheme transaction uses read, write and range locks.

2.3 Replication

Systems supports replication if copies of data items are stored at many distinct sites (Date, 2000). Replication enhances system performance when clients can operate with local copies. Replication means also better availability, data will be available for processing so long as at least one copy of it remains available. Updates of the data items needs to be propagated to all replica copies (Date, 2000). Abadi describes three possible ways to replicate data: the system sends data updates to all replicas at the same time, to an agreed-upon master node first, or to a single (arbitrary) node first (Abadi D. J., 2012).
1. In update all replicas at the same time case consistency cannot be guaranteed without agreement protocol like Paxos, which causes latency.

2. In update agreed-upon master node first case the master node resolves the updates and replicates them to slave nodes either synchronously or asynchronously. It is possible to use mixture mode in which some slave nodes are updated synchronously and some asynchronously. Synchronous replication is consistent, but increases latency. In asynchronous replication consistency depends on how systems handles reads. If reads are directed to master node consistency will hold, but latency increases due to remote clients and overloading of the master node.

3. In update single (arbitrary) node first case the master node is not the same node on every update. Additionally there can be simultaneous updates, which are initiated with different master nodes. If replication is synchronous then latency problems are similar to case 2 with synchronous replication. Simultaneous updates to same data items from different locations causes additional latency problems. If replication is asynchronous then consistency problems like in case 1 and case 2 with asynchronous replication are present.

2.4 Concurrency control

Databasesystems can drive multiple transactions. These transaction can access same data items at the same time. Some kind of mechanism to handle these concurrent accesses are needed. Concurrency control is large topic and some aspects like transaction management and isolation are already discussed in this paper. In this chapter concurrency control mechanism will be discussed very briefly. Bernstein & Goodman lists 47 concurrency controls algorithms and describes 25 of them in detail (Bernstein & Goodman, 1981). In this paper describes two most intresting ones in CAP point of view, locking and multiversion concurrency control.

Traditionally concurrency control mechanism is based on locking. Transaction can issue write and read locks on data items. Locking granularity tells how big portion on database is locked. It can vary from entire database down to table, row, or single data item in a row. Locking solves concurrent update problems and consistency problems (Date, 2000).

In multiversion database system each write on a data item produces new copy of that data item. Data items are never overwritten (Bernstein & Goodman, 1981). Multiversion concurrency control enables reads for older versions of data items and recovery using version history (Palmer, 2013).
3. **CAP Theorem**

CAP theorem was first introduced by Eric Brewer in 2000. It was proven by Gilbert and Lynch in 2002. CAP theorem states that from three properties: Consistency, Availability and Partition tolerance only two can be achieved. All three cannot be achieved simultaneously (Brewer, 2000). In this chapter an overview of CAP theorem is presented and the most common sacrifice is discussed. Also weaker consistency models are be discussed.

### 3.1 Overview of CAP theorem

As figure 5 shows available choices are consistency and availability CA, availability and partition tolerance AP or consistency and partition tolerance CP. In this chapter these properties and available choices shall be studied. Also weaker consistency models eventual consistency and causal consistency will be discussed in this chapter.

![Figure 3 CAP Theorem](image)

**Availability:** In CAP theorem availability means 100% availability. Availability is considered to be one of key success factors for global large scale web applications like webstores. Bad availability leads to poor user experience and customers soon changes to another webstore (Lee & Lin, 2005). Abadi equals availability with latency. Long latency in service makes the service effectively unavailable for user. Even latency of 100 milliseconds affects user experience so that user is not likely to visit again (Abadi D. J., 2012).

**Consistency:** Consistency in CAP theorem is defined somewhat differently than ACID consistency. In CAP context consistency means that reads from different sites result
always same data, so it actually means linearizability (Klepmann, 2014). Gilbert & Lynch: “Atomic ..., or linearizable ..., consistency is the condition expected by most web services today. Under this consistency guarantee, here must exist a total order on all operations such that each operation looks as if it were completed at a single instant. This is equivalent to requiring requests of the distributed shared memory to act as if they were executing on a single node, responding to operations one at a time”. Figure 6 depicts example of non linearizable consistency. In fact CAP consistency is similar to isolation property of ACID.

![Diagram](image)

**Figure 4 Example of non linearizable transaction**

**Partition tolerance:** Partition tolerance means that the system works fine even if there are some network partitions. Network partitions are formed when network connection are broken so that parts of the network cannot be reached. Distributed database system is partitioned when some of its nodes cannot be reached due to network failure. Desired goal in presence of partition is that all partitions would continue functioning (Stonebraker, 2010). Single node failure is not a partition. On CAP point of view a partition needs to have at least one live node in it (Hale, 2010). Figure 6 illustrates network partition.

According to Abadi, CAP states that it is not only partition tolerance that necessitates the trade-off between consistency and availability, but combination of partition tolerance and the actual existence of network partition. CAP allows full consistency and availability unless there is actual network partition (Abadi D. J., 2012).
As CAP theorem states only two of the above mentioned properties can be chosen. From figure 5 can be seen that the choices are CP – partition tolerance and consistency, CA – consistency and availability and AP availability and partition tolerance.

**CA Consistency and Availability**

This would mean that partition tolerance is forfeited. This is clearly not an option in scalable distributed database system (Pritchett, 2008; Hale, 2010). This might work on LAN based database system. Even there, if you need to add another database server or do some maintenance work, the system will not be available at that time. According to Brewer CA can be achievable if network partition is less likely to happen than other kind of major failure.

**CP Consistency and Partition tolerance**

This means that system is consistent and partition tolerant, but not available in presence of network partition. Since CP systems are available when there is no network partition, CA and CP are overlapping. (Abadi D., 2010).

**AP Availability and Partition tolerance**

AP systems are constantly available despite of network partitions, but not necessarily linearizable consistent.

### 3.1.1 Weak consistency models

Strong consistency models like linearizable or sequential consistecy guarantees transparent single-copy image for clients in expence of latency and availability (Fekete & Ramamritham, 2010). To offer better performance and availability some weaker consistency models have been proposed. In this chapter eventual consistency and causal consistency model shall be studied. These consistency models are interesting in CAP-point of view. According to Bailis and Ghodsi these consistency models provide availability.
even in presence of network partition. Other weak consistency model are e.g. Read-your-writes consistency, monotonic read consistency and monotonic write consistency. Those are not studied in detail in this paper.

**Eventual consistency**

CAP theorem has been used as an excuse to weaken consistency in distributed scalable applications. Eventual consistency guarantees that if no updates are coming to a data item, all reads to that data item will eventually return the latest updated value. If all update activity stops, after some time all database servers will converge to the same state. However during this converging period the system may provide arbitrary values. So eventual consistency is rather weak property. Eventual consistency does not provide any safety properties (“nothing bad happens”) it purely liveness property (“something good will eventually happen”) (Bailis & Ghodsi, 2013.)

The information exchange process needed to converge is called anti-entropy. There are variety of anti-entropy protocols. One simple solution is asynchronous all-to-all broadcasts. When a server receives an update, it immediately responds to the client and in the background it sends the update to all other replicas. If there is concurrent updates to the same data item, replicas must choose the winning value. Common method to choose the winning value is “last write wins” (Bailis & Ghodsi 2013.)

Eventual consistency does not guarantee that servers see the updates in correct order. This makes implementing eventually consistent difficult (Lloyd; Freedman; Kaminsky; & Andersen, 2014). Design pattern developed for eventual consistency programming is called ACID 2.0. Associativity: functions can be applied in any order, Commutativity: function arguments can be in any order, Idempotence: function can be called with same input any number of times and D is just a placeholder for Distributed. Applying these design pattern logical monotonicity can be achieved (Bailis & Ghodsi, 2013). The wrong order of updates may appear to users as anomalies.

**Causal consistency**

Causal consistency is the strongest consistency model available in presence of network partitions. (Bailis; Ghodsi; Hellerstein; & Stoica, 2013). In causal consistency model reads must obey partial order happens-before relation. If events do not have happens-before relation they are said to be concurrent. Partial order happens-before relation has three rules: (Lamport, 1978), (Lloyd & al, 2014) and (Elbushra & Lindström, 2015):

1. If operations a and b are executed in same thread of execution and a is executed before b then a→b.
2. If operations a is a send operation in one thread and b is a receive operation of that message in another thread, then a→b.
3. Transitivity: if a→b and b→c, then a→c.

Causal consistency ensures that operations appear in order by these rules, which simplifies software development and eliminates many anomalies seen with eventual consistency (Lloyd & al., 2014). According to Pritchett requiring correct ordering is unnecessary, expensive to support and gives false sense of security (Pritchett, 2008).
3.2 The Most Common sacrifice

After Eric Brewers conjecture on CAP and its proof by Gilbert & Lynch there was some years of hype for forfeiting consistency and ACID properties for availability – especially in industry and NoSql movement. The academics were mostly still. In recent years however, there have arisen some critical voices - even Brewer himself in his “Twelve Years After” – paper. This chapter shall clarify what is the most common of the CAP properties to be forfeited. Ideas and concepts presented in previous chapters are used as a base in this chapter.

3.2.1 Partition tolerance

When would it be possible to forfeit partition tolerance? According to Brewer it is sensible to forfeit partition tolerance if the probability of partition is less than probability of other major disasters. Due to improvement in infrastructure nowadays network partitions are very rare (Brewer, 2012), (Abadi D. J., 2012) and (Stonebraker, 2010).

Designers should mitigate the effect of partitions and keep both availability and consistency. This involves three steps, first systems must detect partition, second step is to enter to partition mode and after partition is healed, system performs recovery actions. Figure 10 illustrates partition detection and recovery.

![Figure 6 Partition mode](image)

When system enters partition mode it needs to decide if it can continue and if so, what measures should it take to enable recovery after partition ends. In quorum based environment if systems resides on node with quorum, it can probably continue normal operations. If it does not have quorum it must stop. Vogels points out that in some applications like Amazon shopping chart it is not acceptable that any partition stops working. In such case partitions choose new set of nodes and continue working (Vogels, 2008). The 2 of 3 in CAPS is misleading, you should not abandon availability or consistency in favor of partition tolerance, but you should design your system so that in presence of partition it takes proper actions in order to be able recover itself after the partition heals (Brewer, 2012.)

People in NoSql movement think otherwise. Many, like Amazon CEO Werner Vogels takes partition tolerance as granted and are ready to give up either consistency or availability (Vogels, 2008). You cannot forfeit partition tolerance in distributed system (Pritchett, 2008) and (Hale, 2010).
3.2.2 Availability

Traditional RDBMS systems with strong ACID properties tend to forfeit availability in favor for consistency in presence of partition (Brewer, 2012). Abadi equals availability with latency. If user experienced latency is more than a few seconds the system can be considered to be not available. Amazon has reported that only 100 milliseconds extra latency decreased sales 1%. Google has reported 25 % revenue loss due to 500ms (from 400ms to 900ms) increase in displaying search results. Microsoft Bing has shown that 2 seconds slowdown queries/user by -1.8% and revenue/user by -4.3% (Wang;Sun;Deng;& Huai, 2013). So availability and latency are important factors to consider in web application development.

Even modern banking systems fear forfeiting availability, because higher availability means higher revenues. If an ATM system detects partition it may set up a lower limit for account balance, but allows withdrawals as long as account balance stays above the limit (Brewer, 2012).

Availability can be forfeited in applications that user has to use sometimes, like some office systems, e.g hour reporting software. If the system is not available users are advised to store their data elsewhere and input it to the system after it is online again. If users can choose, she will choose system with better availability and lower latencies (Lee & Lin, 2005).

In literature forfeiting availability is not a widely discussed subject. Mainly industrial sources warn that if you forfeit availability you will be soon out of business.

3.2.3 Consistency

To forfeit consistency or not is definitely the most debated subject around CAP theorem in literature and in industry. After the introduction of CAP theorem NoSql databases using weak consistency model emerged. Main motivation in loosening consistency was the 2 of 3 statement of CAP, systems should be tolerant to network partitions and available all the time (Anderson, & al, 2015), (Bailis & Ghodsi, 2013) and (Pritchett, 2008). According to Abadi, Brewer and Stonebraker this is however a misinterpretation of CAP theorem. In normal situation, when there is no partition, consistency should not be forfeited lightly.

For application developer forfeiting consistency means that safe ACID properties are no longer provided by DDBMS. Anomalies caused by weak consistency models has to be dealt in application code. When applying weak consistency developers should search opportunities to relax consistency by splitting ACID transaction to smaller units, use message queues to avoid 2PC, use idempotent and commutative operations (Pritchett, 2008) and (Bailis & Ghodsi, 2013). If something goes wrong system should be able to compensate errors. Compensation may be external to the system e.g bank charges overdraft fee and repayment in case account value is below zero due to overdraft from ATM machines (Brewer, 2012). Developer needs to maximize benefit of weak consistency B against cost of inconsistency anomalies C and rate of anomalies R. (Bailis & Ghodsi, 2013).

Maximize B-RC
Despite this weakness, eventually consistent databases have proven to be useful. For example, a social networking service using globally distributed datacenters often use eventual consistency. Results from PBS tools show that eventual consistency often provides consistent data in tens of milliseconds (Bailis & al., 2012).

In table 1 main consistency models are compared. It shows that stronger consistency models lead to poor availability and performance. Weaker consistency models provide better availability and performance, but makes application development difficult.

<table>
<thead>
<tr>
<th>Model</th>
<th>Consistency</th>
<th>Availability</th>
<th>Performance</th>
<th>Application development</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serializability</td>
<td>Strong</td>
<td>Weak</td>
<td>Weak</td>
<td>Easy</td>
</tr>
<tr>
<td>Causal</td>
<td>Average</td>
<td>Good</td>
<td>Average</td>
<td>Average</td>
</tr>
<tr>
<td>Eventual</td>
<td>Weak</td>
<td>Good</td>
<td>Good</td>
<td>Hard</td>
</tr>
</tbody>
</table>

*Table 1 Consistency model summary*

### 3.3 Discussion

Consistency is the most discussed topic in literature. NoSql movement and many practitioners in global web market companies like Dan Pritchett of Amazon are openly in favor to at least weaker consistency models. Some scholars are concerned that CAP theorem has been interpreted wrong way if it is used to justify weakening consistency in non-partitioned system. Consistency should not be forfeited when there is no network partition. Only in presence of partition system should choose either availability or consistency.

Forfeiting availability or long latencies cast money to many businesses. “Keep money making machine alive” as Julian Browne puts it. Availability really cannot be forfeited in distributed systems where users can vote by clicking away from service.

Also partition tolerance cannot be forfeited in distributed system. If possibility of network partition is not taken account for in system design, both availability and consistency are in danger when partition occurs.

Recent development in network infrastructure and protocols have made network partitions extremely rare even in WAN environment. Also new distributed high availability transaction management systems like CALVIN (Thomson, Diamond, Weng, Ren, Shao, Abadi, 2014) and HAT (Bailis;Fekete;Ghodsi;Hellerstein;& Stoica, 2013) are emerging. This development is diminishing the meaning of original CAP-theorem and emphasising latency vs. consistency trade-off. (Abadi D. J., 2012). These new view points will be discussed in the next chapter.
4. Modern viewpoint

CAP theorem was first introduced in late 1990’s. Since then there has been much development in network infrastructure. CAP tradeoffs are applicable only for partitioned system, they do not apply for normal behavior of the system, where network partition is not present. In normal situation there is tradeoff between consistency and latency. Abadi suggest PACELC tradeoffs instead: does system tradeoff under Partitions Availability or Consistency, or Else in normal situation does it tradeoff Latency or Consistency (Abadi, 2012). In PACELC classification there would be six options:CA/EL, CA/EC, CP/EL, CP/AP, AP/EL or AP/EC.

CAP addresses only a fraction of all possible faults that affect distributed database systems. Most failure situtations, including network partition in LAN network can be easily recovered by lots of algorithms. Nowadays also WAN infrastrucute is so advanced that network partition is extremely rare. (Stonebraker, 2010). This reduces the value of CAP theorem. Many of these new proposals put weight more on PACELC tradeoffs and scalability rather than pure CAP tradeoffs.

New ideas about how to handle concurrency control and distributed transactions has emerged. In these new proposals network partition is seen more like any other type of failure which the system has to deal with. In this chapter these new methods will be introduced and their impact on CAP theorem will be studied.

4.1 Proposals for distributed concurrency control and transaction handling

Traditional RDBMS offer easy to use transactional semantics for application developers. Yet they suffer from poor availability (Bailis & al., 2013). NoSql database systems offer high availability but application development with them is difficult due to lack of proper transaction semantics and loose consistency models. (Bailis &al., 2013, Baker & al., 2011, Zhu & Wang, 2015, Kraska & al, 2013). In this chapter some proposals for highly available transaction handling and concurrency control will be discussed.

Megastore is Google’s data store, which provides strong consistency guarantees and high availability (Baker & al., 2011). It is based on Google’s Bigtable key – value database. Data is replicated using low latency implementation of Paxos protocol across datacenters. Data is partitioned to entity groups that span across datacenters. ACID consistency is guaranteed within entity group. Figure 9 depicts Megastore’s replication and partition scheme. Megastore uses multiversion concurrency control and supports current, snapshot and inconsistent reads. Two phase commit in transactions that access data in multiple entity groups is supported, but not encouraged, because it has higher latency and risk of contention. Using queues is encouraged instead.
Megastore’s Paxos implementation consists of application server, megastore library, replication server and a simple coordinator services in all replicas. There are also witness replicas, which do not actually store any data and have only replication server. They participate in Paxos votes and writing to write ahead logs. They are used when there is not enough full replicas to form a quorum. There are also read-only replicas, which do not participate in Paxos, they only contain snapshot of the data. Megastore’s Paxos implementation causes limitation to scalability because it allows only one transaction per entity group at a time (Kraska; Pang; Frankling; Madden; & Fekete, 2013).

Network partitions or other failures in replica are detected by coordinators using Chubby locks (Burrows, 2006). When a problem with a replica is detected, a number of measures can be taken. Users can be rerouted to healthy replicas, if they can be reached. Replica’s coordinators can be disabled or entire replica can be disabled.

**Multi Datacenter Consistency** MDCC (Kraska & al., 2013) guarantees read committed isolation level. MDCC uses Generalized Paxos for replication across multiple data partitions. It requires only one network round trip in common case. MDCC achieves strong consistency in wide–area network with similar costs to eventual consistency. MDCC’s commit protocol tolerates datacenter failures without compromising consistency. MDCC architecture consist of record manager, DB library and storage servers. DB Library is responsible for coordinating of the replication, consistency of the data and determines the outcome of the transaction. DB Library can be deployed by an application server.

**SHAFT** (Zhu & Wang, 2015) provides strong serializable consistency even on failures. It is also based on Paxos protocol. It uses two Paxos instances, processing instance and decision instance. Processing instance is responsible of leader election, consensus and majority. Decision instance makes it possible for client to abort ongoing transaction. Fault tolerance in SHAFT is achieved by concurrent transactions ability to detect and heal leader failures in other concurrent transactions. Leader of a transaction can take place of failed leader of another transaction and drive that transaction to conclusion. Serializability in SHAFT is achieved by two phase locking protocol.

Thomson introduced (Thomson, et al., 2014) a fast distributed database transaction service called **CALVIN**. In CALVIN transaction processing and concurrency control is separated from physical database. It has three layers, sequencer, scheduler and database layers. These layers are distributed to all partition in every replicas of the system. Each sequencer takes transaction requests from clients during a time frame of 10ms called an epoch. It then
compiles these transactions into a batch. These patches are then replicated to global transaction sequence. Every sequencer forwards transaction patches to schedulers who can access records involved in transaction. Sequencer can use asynchronous or Paxos based synchronous replication schemes. Asynchronous replication offers better performance, but recovering from failure on master replica is very complex. Schedulers are responsible of concurrency control using deterministic locking protocol. For application developers CALVIN provides developer friendly server side transactions.

**HATs** Highly Available Transactions (Bailis;Fekete;Ghodsi;Hellerstein;& Stoica, 2013) provides many of ACID properties like atomicity, Isolation levels read committed and repeatable read. HATs do not guarantee serializability or global database integrity rules.

### 4.2 Effect on CAP theorem

CAP theorem addresses that strict serializable consistency and availability cannot be achieved in distributed database system in the presence of network partition (Gilbert & Lynch, 2012). However previously introduced proposals show that ACID transactions can be used and at least read committed isolation level can be achieved while maintaining availability in presence of network partition. Many traditional RDBMSs provides read committed isolation level as their default configuration (Bailis. Fekete, Ghodsi, Hellerstein & Stoica, 2013).

Application development with eventual consistency and absence of transaction semantics is very difficult. All the new proposals offer familiar transaction handling framework for application developers to lessen the burden of application development in eventually consistent environment.
5. Conclusions

This paper has discussed properties of distributed database systems. In distributed database data is spread among servers in many locations connected via communication network. Distribution boosts performance, when mostly used data can be located near user. Users have seamless access to data in remote locations. In user point of view using distributed database should not be different than using local database. Complexities due to data distribution were discussed in chapter 2 including transaction management, concurrency control and some replication schemes.

CAP theorem was introduced in chapter 3. Distributed database cannot achieve all 3 properties, Partition tolerance, Availability and Consistency at the same time. This conjecture was introduced by Eric Brewer in late 1990’s and was formally proven by Gilbert and Lynch in 2002. Soon after the theorem was introduced NoSql database movement started to use it as excuse to use weaker consistency models. Recently it has been discovered that network partition are rare and the real trade off in healthy distributed database system is between latency and consistency (Abadi D., 2010). In that respect weak consistency models and PACELC trade off model was also discussed in chapters 3 and 4.

In chapter 3 CAP theorem was further discussed in light of research question “what is the most common sacrifice according to distributed database literature”. Partition tolerance cannot be really forfeited in distributed database. System behavior in presence of partition should be carefully planned, otherwise both availability and consistency are in danger when network partition occurs. Poor availability costs money to businesses in means of latency and bad user experience. Availability can be forfeited only in systems where user have no choice, but to retry, like some office systems.

New proposals for distributed transaction handling and concurrency control discussed in chapter 4 enhance performance, scalability and availability of distributed databases. They also lessen the difficulties in application development for eventually consistent databases.

NoSql community has been in favor of availability over consistency. There are also some measurements available which indicates that eventual consistency provides fairly good results with low latencies.
References


Baker, Jason; Bond, Chris; Corbett, James C; Furman, JJ; Khorlin, Andrey; Larson, James; Léon, Jan-Michel; Li, Yawei; Lloyd, Alexander; Yushprakh, Vadim; Google Inc. (2011). *Megastore:Providing Scalable, Highly Available Storage for Interactive Services. 5th Biennial Conference on Data Systems Research (CIDR’11)*, (s. 223). Asilomar.


Appendix A. Resources and links


Website about high scalability: http://highscalability.com/

Transaction Processing and the Barbarian Hordes: https://onedrive.live.com/?id=84F3C5EF51D06E8B!164&cid=84F3C5EF51D06E8B