# CAP Theorem: Revision of its Related Consistency Models

Francesc D. Muñoz-Escoí*
Rubén de Juan-Marín
José-Ramón García-Escrivá
*Instituto Universitario Mixto Tecnológico de Informática,*
*Universitat Politècnica de València, 46022 Valencia (Spain)*

J. R. González de Mendívil
*Depto. de Ingeniería Matemática e Informática,*
*Universidad Pública de Navarra, 31006 Pamplona (Spain)*

José M. Bernabéu-Aubán
*Instituto Universitario Mixto Tecnológico de Informática,*
*Universitat Politècnica de València, 46022 Valencia (Spain)*

**Abstract**

The CAP theorem states that only two of these properties can be simultaneously guaranteed in a distributed service: (i) consistency, (ii) availability, and (iii) network partition tolerance. This theorem was stated and proved assuming that "consistency" refers to atomic consistency. However, multiple consistency models exist and atomic consistency is located at the strongest edge of that spectrum.

Many distributed services deployed in cloud platforms should be highly available and scalable. Network partitions may arise in those deployments and should be tolerated. One way of dealing with CAP constraints consists in relaxing consistency. Therefore, it is interesting to explore the set of consistency models not supported in an available and partition-tolerant service (CAP-constrained models). Other weaker consistency models could be maintained when scalable services are deployed in partitionable systems (CAP-free models). Three contributions arise: (1) multiple other CAP-constrained models are identified, (2) a borderline between CAP-constrained and CAP-free models is set, and (3) a hierarchy of consistency models depending on their strength and convergence is built.

KEYWORDS: Inter-replica consistency; CAP theorem; Service availability; Network partition; Consistency model

## 1 Introduction

Scalable distributed services try to maintain their service continuity in all situations. When they are geo-replicated, a trade-off exists among three properties: replica consistency (C), service availability (A) and network partition tolerance (P). Only two of those three properties can be simultaneously guaranteed. Such trade-off was suggested long time ago (Davidson et al., 1985) [1], explained by Fox and Brewer [2] in 1999 and proved by Gilbert and Lynch [3] in 2002. However, the compromise between strongly consistent actions, availability and tolerance to network partitions was already implicit in Johnson and Thomas (1975) [4] and justified by Birman and Friedman [5] in 1996.

Service availability and network partition tolerance are dichotomies. They are either respected or not. Service availability means that *every client request* that reaches a service instance should be

*e-mail: *fmunyoz@iti.upv.es*

answered. When a network partition arises, the instances of a service may be spread among multiple disjoint node subgroups. Network partition tolerance means that every service instance subgroup goes on while the network remains partitioned.

On the other hand, service replica consistency admits a gradation of consistency levels. In spite of this, when we simply refer to "consistency" we understand that it means atomic consistency [6]; i.e., that all instances are able to maintain the same values for each variable at the same time, providing a behaviour equivalent to that of a single copy. This led to assume that kind of consistency in the original proofs of the CAP theorem [3].

With the advent of cloud computing, it is easy to develop and deploy highly scalable distributed services [7]. Those applications usually provide world-wide services: they are deployed in multiple datacentres and this implies that network partition tolerance is a must for those services. Thus, those services regularly prioritise availability when they should deal with the constraints of the CAP theorem, and consistency is the property being sacrificed. However, that sacrifice should not be complete. Brewer [8] explains that network partitions are rare, even for world-wide geo-replicated services. If services demand partition tolerance and availability, their consistency may still be quite strong most of the time, relaxing it when any temporary network partition arises.

It is worth exploring which levels of consistency are strong enough to be directly implied by the CAP constraints; i.e., those *CAP-constrained* models are not supported when the network becomes partitioned. On the other hand, there are several relaxed models that remain available when a network partition arises. They constitute the *CAP-free* set of models and there is a (not yet completely known) frontier between CAP-free and CAP-constrained models. Two questions arise in this scope: (1) Does CAP affect only to atomic consistency or are there any other "CAP-constrained" models? (2) If there were any other models, what would the CAP-constrained vs. CAP-free frontier be? Although some partial answers to these questions have been given in previous papers [9, 10, 11], let us provide a revised answer to them in the following sections.

# 2 System Model

A distributed system $S = (P, O)$ is assumed. The real-time domain is represented by set $T$. $S$ is partially synchronous and consists of: (1) a set of processes $P$ connected by a network where processes communicate through message passing, and (2) a set of objects $O$, with their states and methods. Processes in $P$ may fail. Scalable distributed services may be deployed in $S$. Those services consist of a set of objects $O$. Objects are replicated in order to improve their availability. Their instances are deployed in $P$ using a replication protocol and respecting some replica consistency model.

Function $Connect : P \times P \times T \rightarrow \{false, true\}$, used as $Connect(p_1, p_2, t)$, returns *true* when processes $p_1$ and $p_2$ are connected at time $t$, and *false* otherwise. Communication may fail when a temporary network partition occurs, defined as follows.

**Definition 2.1** (Network partition)**.** *When a network partition $NP = (S, K, it, et)$ occurs in a system $S = (P, O)$ from some initial time $it \in T$ to an end time $et \in T$ ($it < et$), $S$ becomes partitioned in a set $K$ of network components, with $\mid K \mid > 1$, such that:*

1. $\bigcup_{i \in K} S_i \subseteq S$, *where* $S_i = (P_i, O)$
2. $\bigcup_{i \in K} P_i \subseteq P$
3. $\forall i, j \in K, i \neq j : P_i \bigcap P_j = \emptyset$
4. $\forall i, j \in K, i \neq j, \forall p_m \in P_i, \forall p_n \in P_j, \forall t \in T, it \leq t \leq et : Connect(p_m, p_n, t) = false$
5. $\forall i \in K, \forall p_m, p_n \in P_i, \forall t \in T, it \leq t \leq et : Connect(p_m, p_n, t) = true$

Processes in different components cannot communicate with each other. Processes in the same component intercommunicate without problems. A *partitionable* system model is assumed in regard to process behaviour.

**Proposition 2.1** (Partitionable system)**.** *When a network partition $NP = (S, K, it, et)$ occurs in $S = (P, O)$, every operation from every process $p_i \in P$ is able to start and/or finish in a regular way in the $(it, et)$ interval, independently on the connectivity of $p_i$ with each other process $p_j \in P$.*

According to Prop. 2.1, no operation gets indefinitely blocked while a network partition lasts in $S$. Considering the CAP constraints, availability and network partition tolerance are respected, while consistency compliance may be sacrificed.

# 3 Basic Specification

Viotti and Vukolić propose a framework for specifying distributed (non-transactional) data consistency models in [12], based on that presented in [13, 14]. Since the CAP theorem involves software services deployed in distributed systems, it makes sense to consider those models in this scope. That framework may be summarised as follows.

## 3.1 Specification Framework

Services consist of processes and objects. Object values belong to set $V$. Processes interact with objects invoking their operations, whose types belong to set $OT$.

Tuples *(proc,type,obj,ival,oval,st,rt)* represent operations, where:

- $proc \in P$ is the identifier of the process that invokes the operation.
- $type \in OT$ is the operation type; e.g., *wr* for writes and *rd* for reads.
- $obj \in O$ is the identifier of the invoked object.
- $ival \in V \cup \{\sqcup\}$ is the operation input value, or $\sqcup$ in case of a read operation.
- $oval \in V \cup \{\sqcup, \nabla, \Theta\}$ is the operation output value, or $\sqcup$ in case of a write or $\nabla$ if the operation does not return or $\Theta$ when a write completes in *proc* but not in other subsets of $P$.
- $st \in T$ is the operation invocation (i.e., start) time.
- $rt \in T$ is the operation return time.

In a tuple $T = (e_1, \ldots, e_n)$, $T.e_i$ refers to element $e_i$ in that tuple.

A history $H$ is a set of operations. A history contains all operations invoked in an execution $E$ of $S$. $H\,|_{wr}$ (respectively, $H\,|_{rd}$) denotes the set of write (respectively, read) operations in a history $H$. Formally, $H\,|_{wr} = \{op \in H : op.type = wr\}$.

The following relations are needed: (1) *rb* (*returns-before*) is a partial order on H based on real-time precedence: $rb \equiv \{(a,b) : a,b \in H \wedge a.rt < b.st\}$, (2) *ss* (*same-session*) is an equivalence relation on H that groups the operations invoked by the same process: $ss \equiv \{(a,b) : a,b \in H \wedge a.proc = b.proc\}$, (3) *so* (*session order*) is a partial order defined as: $so \equiv rb \cap ss$, (4) *ob* (*same-object*) is an equivalence relation on H that groups the operations invoked on the same object: $ob \equiv \{(a,b) : a,b \in H \wedge a.obj = b.obj\}$, and (5) *concur* is a symmetric binary relation that includes all pairs of real-time concurrent operations invoked on the same object: $concur \equiv ob \setminus rb$.

Moreover, there are other specification aspects to be considered. To begin with, the *concur* relation is complemented with a function $Concur : H \to 2^H$ that denotes the set of write operations concurrent with a given operation: $Concur(a) \equiv \{b \in H\,|_{wr} : (a,b) \in concur\}$. The projection $rel\,|_{wr \to rd}$ identifies all pairs of operations in relation *rel* that consist of a write and a read operation. $H/\approx_{rel}$ denotes the set of equivalence classes determined by relation *rel*, $rel^{-1}$ denotes the inverse relation of *rel* and $rel(a) = \{b \in A : (a,b) \in rel\}$. Note that $rel(a)$ is a set, since there may be many elements related transitively to $a$.

An execution is defined as $E = (H, vis, ar)$ and is built on a history $H$, complemented with two relations *vis* and *ar* on elements of $H$, where: (1) *vis* (*visibility*) is an acyclic partial order that accounts for the propagation of write operations; two write operations are *invisible* to each other when they are

Table 1: Definition of basic consistency predicates.

| **Predicate** | **Definition** |
|---|---|
| RVAL($\mathscr{F}$) | $\forall op \in H : op.oval \in \mathscr{F}(op, cxt(E,op))$ |
| PRAM | $so \subseteq vis$ |
| SINGLEORDER | $\exists H' \subseteq \{op \in H : op.oval = \nabla\} : vis = ar \setminus (H' \times H)$ |
| LAZYSINGLEORDER | $\exists H' \subseteq \{op \in H : op.oval \in \{\nabla, \Theta\}\} : vis = ar \setminus (H' \times H)$ |
| REALTIME | $rb \subseteq ar$ |
| REALTIMEWRITES | $rb \mid_{wr \to op} \subseteq ar$ |
| SEQRVAL($\mathscr{F}$) | $\forall op \in H : Concur(op) = \emptyset \Rightarrow op.oval \in \mathscr{F}(op, cxt(E,op))$ |
| EVENTUALVISIBILITY | $\forall a \in H, \forall [f] \in H/\approx_{ss} : \mid \{b \in [f] : (a,b) \in rb \wedge (a,b) \notin vis\} \mid < \infty$ |
| NOCIRCULARCAUSALITY | $acyclic(hb)$ |
| STRONGCONVERGENCE | $\forall a,b \in H \mid_{rd} : vis^{-1}(a) \mid_{wr} = vis^{-1}(b) \mid_{wr} \Rightarrow a.oval = b.oval$ |
| CAUSALVISIBILITY | $hb \subseteq vis$ |
| CAUSALARBITRATION | $hb \subseteq ar$ |
| TIMEDVISIBILITY($\Delta$) | $\forall a \in H \mid_{wr}, \forall b \in H, \forall t \in T : a.rt = t \wedge b.st = t + \Delta$ $\Rightarrow (a,b) \in vis$ |
| REALTIMEWW | $rb \mid_{wr \to wr} \subseteq ar$ |
| CONCURRVAL($\mathscr{F}$) | $\forall op \in H : op.oval \in \mathscr{F}(op, cxt(E,op) \cup Concur(op))$ |
| K-REALTIMEREADS(K) | $\forall a \in H \mid_{wr}, \forall b \in H \mid_{rd}, \forall PW \subseteq H \mid_{wr}, \forall pw \in PW : \mid PW \mid < K \wedge$ $(a,pw) \in ar \wedge (pw,b) \in rb \wedge (a,b) \in rb \Rightarrow (a,b) \in ar$ |
| NOJOIN | $\forall a_i, b_i, a_j, b_j \in H : a_i \not\approx_{ss} a_j \wedge (a_i,a_j) \in ar \setminus vis \wedge a_i \preceq_{so} b_i \wedge$ $a_j \preceq_{so} b_j \Rightarrow (b_i,b_j), (b_j,b_i) \notin vis$ |
| ATMOSTONEJOIN | $\forall a_i, a_j \in H : a_i \not\approx_{ss} a_j \wedge (a_i,a_j) \in ar \setminus vis \Rightarrow \mid \{b_i \in H : a_i \preceq_{so} b_i \wedge$ $(\exists b_j \in H : a_j \preceq_{so} b_j \wedge (b_i,b_j) \in vis)\} \mid \leq 1 \wedge \mid \{b_j \in H : a_j \preceq_{so} b_j$ $\wedge (\exists b_i \in H : a_i \preceq_{so} b_i \wedge (b_j,b_i) \in vis)\} \mid \leq 1$ |
| PEROBJECTPRAM | $(so \cap ob) \subseteq vis$ |
| PEROBJECTSINGLEORDER | $\exists H' \subseteq \{op \in H : op.oval = \nabla\} : vis \cap ob = ar \cap ob \setminus (H' \times H)$ |

not ordered by *vis*, and (2) *ar* (*arbitration*) is a total order on operations of the history that specifies how conflicts due to invisible operations are resolved in $E$ in order to respect its consistency models.

The *happens-before* (*hb*) partial order is defined as the transitive closure of the union of *so* and *vis*; i.e., $hb \equiv (so \cup vis)^+$.

Some extensions to [12] are needed in order to deal with partitionable networks. Those extensions are specified hereafter.

$\mathscr{E}$ is the set of executions in $S$. $\mathscr{E}_P$ is the subset of $\mathscr{E}$ that contains all executions in which the conditions of Def. 2.1 are met, i.e., their network becomes temporarily partitioned. On the other hand, $\mathscr{E}_C$ is the complementary subset of $\mathscr{E}_P$ in which no network partition has occurred. Thus, $\mathscr{E} = \mathscr{E}_P \cup \mathscr{E}_C$ and $\mathscr{E}_P \cap \mathscr{E}_C = \emptyset$.

The context $C$ of an operation *op* in execution $E$ is defined as: $C_{op} = cxt(E,op) \equiv (E.vis^{-1}(op), E.vis \mid_{C_{op}.H}, E.ar \mid_{C_{op}.H})$, i.e., a projection of $E$ that only keeps in its history those operations in $vis^{-1}(op)$. For each data type, function $\mathscr{F}$ specifies the set of intended return values of *op* in relation to its context: $\mathscr{F}(op, cxt(E,op))$. With $\mathscr{F}$, the *return value consistency* is defined as: RVAL($\mathscr{F}$) $\equiv \forall op \in E.H : op.oval \in \mathscr{F}(op, cxt(E,op))$. In this scope, we use by default a *register* data type ($\mathscr{F}_{reg}$). Let us explain how *op.oval* is chosen from $C_{op}$ in $\mathscr{F}_{reg}$. From $vis^{-1}(op)$, only those $op_2 \in C_{op}.H : op_2.oval \notin \{\nabla, \Theta\} \wedge op_2.obj = op.obj$ are considered. Multiple candidates may arise. If so, only those operations without *vis*-successors in $C_{op}.H$ are assessed. From that subset, with operations invisible to each other, the read value is that of the latest operation in *ar* order. If no candidate exists, then *op.oval* is a special value $\perp$.

Let us use an execution $E_x$ for explaining the specification aspects presented in previous paragraphs. Let $S$ be $(\{p_1,p_2\},\{x\})$ and $E_x = (\{o_1 = (p_1,wr,x,1,\sqcup,0,1),\ o_2 = (p_2,wr,x,2,\sqcup,0,1),\ o_3 = (p_1,rd,x,\sqcup,1,1,2),\ o_4 = (p_2,rd,x,\sqcup,2,1,2),\ o_5 = (p_1,rd,x,\sqcup,2,3,4),\ o_6 = (p_2,rd,x,\sqcup,1,3,4)\},\ \{(o_1,o_3),$

$(o_3,o_5)$, $(o_2,o_4)$, $(o_4,o_6)$, $(o_2,o_5)$, $(o_1,o_6)\}$, $\{(o_4,o_1),(o_1,o_6),(o_6,o_3),(o_3,o_2),(o_2,o_5)\}$). Local execution order introduces $(o_1,o_3)$, $(o_3,o_5)$, $(o_2,o_4)$ and $(o_4,o_6)$ in *vis*. Values written in $o_1$ and $o_2$ are propagated to the other process, so $(o_2,o_5)$ and $(o_1,o_6)$ are in *vis*. Since *ar* is a total order, it sets this ordering in $E_x$: $o_4 < o_1 < o_6 < o_3 < o_2 < o_5$. There are four reads: $o_3, o_4, o_5$ and $o_6$, with these context histories: $C_{o_3}.H = \{o_1\}$, $C_{o_4}.H = \{o_2\}$, $C_{o_5}.H = \{\underline{o_1}, o_3, o_2\}$, $C_{o_6}.H = \{\underline{o_2}, o_4, o_1\}$. In each $C_i.H$, the underlined operations are discarded when RVAL($\mathscr{F}$) is applied, since they have subsequent operations in *vis* that are also in $C_i.H$. From the remaining subsets, any potential conflict is resolved according to *ar*. This explains the read values.

## 3.2 Distributed Consistency Models

Viotti and Vukolić [12] distinguish ten groups of consistency models: (1) linearisable and other strong models, (2) weak and eventual consistency, (3) PRAM and sequential consistency, (4) session guarantees, (5) causal models, (6) staleness-based models, (7) fork-based models, (8) composite and tunable models, (9) per-object models, and (10) synchronised models. Synchronised models are described in [12] for completeness; they make sense in multiprocessor computers but not in general distributed systems. The models in the eighth group cannot be specified with the proposed consistency predicates. Therefore, no relation with the models contained in other groups can be set for them. Those two groups are not considered hereafter. Table 1 shows a set of consistency predicates. With those predicates, consistency models may be specified as shown in Table 2.

Consistency models are also known as *consistency conditions*. Both terms are synonyms, but generate two different kinds of names. Conditions use nouns (e.g., linearisability [15]) while models use adjectives (e.g., atomic, regular and safe [6]). For the sake of uniformity, this paper uses models and adjectives in order to refer to consistency in all cases.

An execution $E$ satisfies a consistency model $M$ built as a conjunction of multiple consistency predicates ($M \equiv \mathscr{P}_1 \wedge \cdots \wedge \mathscr{P}_n$) iff $E$ satisfies all those predicates. Formally: $E \models M \Leftrightarrow E \models \mathscr{P}_1 \wedge \cdots \wedge \mathscr{P}_n$.

In regard to the consistency models specified in Table 2, PREFIXSEQUENTIAL($\mathscr{F}$) is derived from the "prefix consistency" proposed in Bayou [23]. Bayou manages a partitionable system. To this end, write operations have two states: tentative and committed, that are modelled using $\Theta$ or $\sqcup$, respectively, as the value of the *oval* operation attribute. That management is specified using LAZYSINGLEORDER in Table 2. In Bayou, a write operation $op$ returns control once it reaches a single server $p_i$. At that time, $op$ is still tentative (i.e., $op.oval = \Theta$). To be committed, $p_i$ propagates $op$ to a primary manager. The primary manager for $op.obj$ chooses a commit order (that conditions the *ar* relation in that execution) for all new writes on that object and that chosen sequence is kept in a log and lazily communicated to every other process. Disconnected nodes should eventually contact the primary manager to learn that commit order. At that time, those previously disconnected processes communicate their tentative writes to the primary (to be ordered on the next commit) and apply the already committed writes on their local replicas. This means that tentative writes may be undone and reapplied in their correct sequence position when they had been initially applied in a disconnected node. When a write $op$ is applied onto the replica of object $op.obj$ in a process $p_j$ in the commit order, $op.oval$ becomes $\sqcup$ in the $p_j$'s view of that history. Such view may be represented as $H\,|_{p_j}$. This explains why different processes in the same execution may have different available committed prefixes of that execution at the same time in the PREFIXSEQUENTIAL($\mathscr{F}$) model.

## 3.3 CAP-related Definitions

Let assume that the executions in system $S$ are driven by a consistency model $CM \equiv \mathscr{P}_1 \wedge \cdots \wedge \mathscr{P}_n$. All executions in $\mathscr{E}_C$ comply always with the definition of *CM*. However, that behaviour may vary when network partitions arise. That fact originates the following definitions.

**Definition 3.1** (CAP-free consistency model)**.** *CM is CAP-free if every execution E in $\mathscr{E}_P$ respects all consistency predicates that define CM.*

*Formally:* $\forall E \in \mathscr{E}_P : E \models \mathscr{P}_1 \wedge \cdots \wedge \mathscr{P}_n$.

Table 2: Definition of basic consistency models.

| **Model** | **Ref.** | **Definition** |
|---|---|---|
| *1.- Linearisable and other strong models* | | |
| LINEARISABLE($\mathscr{F}$) | [15] | SINGLEORDER $\wedge$ REALTIME $\wedge$ RVAL($\mathscr{F}$) |
| REGULAR($\mathscr{F}$) | [6] | SINGLEORDER $\wedge$ REALTIMEWRITES $\wedge$ RVAL($\mathscr{F}$) |
| SAFE($\mathscr{F}$) | [6] | SINGLEORDER $\wedge$ REALTIMEWRITES $\wedge$ SEQRVAL($\mathscr{F}$) |
| *2.- Weak and eventual consistency* | | |
| WEAK | | No requirement |
| EVENTUAL($\mathscr{F}$) | [16] | EVENTUALVISIBILITY $\wedge$ NOCIRCULARCAUSALITY $\wedge$ RVAL($\mathscr{F}$) |
| STRONGEVENTUAL($\mathscr{F}$) | [17] | EVENTUAL($\mathscr{F}$) $\wedge$ STRONGCONVERGENCE |
| *3.- PRAM and sequential consistency* | | |
| PRAM($\mathscr{F}$) | [18] | PRAM $\wedge$ RVAL($\mathscr{F}$) |
| SEQUENTIAL($\mathscr{F}$) | [19] | SINGLEORDER $\wedge$ PRAM($\mathscr{F}$) |
| *4.- Session guarantees* | | |
| MONOTONICREADS | [20] | $\forall a \in H, \forall b,c \in H\mid_{rd}: (a,b) \in vis \wedge (b,c) \in so \Rightarrow (a,c) \in vis$ |
| READYOURWRITES | [20] | $\forall a \in H\mid_{wr}, \forall b \in H\mid_{rd}: (a,b) \in so \Rightarrow (a,b) \in vis$ |
| MONOTONICWRITES | [20] | $\forall a,b \in H\mid_{wr}: (a,b) \in so \Rightarrow (a,b) \in ar$ |
| WRITESFOLLOWREADS | [20] | $\forall a,c \in H\mid_{wr}, \forall b \in H\mid_{rd}: (a,b) \in vis \wedge (b,c) \in so \Rightarrow (a,c) \in ar$ |
| *5.- Causal models* | | |
| CAUSAL($\mathscr{F}$) | [21] | CAUSALVISIBILITY $\wedge$ CAUSALARBITRATION $\wedge$ RVAL($\mathscr{F}$) |
| CAUSAL+($\mathscr{F}$) | [22] | STRONGCONVERGENCE $\wedge$ CAUSAL($\mathscr{F}$) |
| REALTIMECAUSAL($\mathscr{F}$) | [9] | REALTIME $\wedge$ CAUSAL($\mathscr{F}$) |
| *6.- Staleness-based models* | | |
| PREFIXSEQUENTIAL($\mathscr{F}$) | [23] | LAZYSINGLEORDER $\wedge$ MONOTONICWRITES $\wedge$ RVAL($\mathscr{F}$) |
| TIMEDCAUSAL($\mathscr{F},\Delta$) | [24] | CAUSAL($\mathscr{F}$) $\wedge$ TIMEDVISIBILITY($\Delta$) |
| TIMEDSERIAL($\mathscr{F},\Delta$) | [25] | SINGLEORDER $\wedge$ TIMEDVISIBILITY($\Delta$) $\wedge$ RVAL($\mathscr{F}$) |
| K-LINEARISABLE($\mathscr{F},K$) | [26] | SINGLEORDER $\wedge$ K-REALTIMEREADS($K$) $\wedge$ REALTIMEWW $\wedge$ RVAL($\mathscr{F}$) |
| K-REGULAR($\mathscr{F},K$) | [26] | SINGLEORDER $\wedge$ K-REALTIMEREADS($K$) $\wedge$ REALTIMEWW $\wedge$ CONCURRVAL($\mathscr{F}$) |
| K-SAFE($\mathscr{F},K$) | [26] | SINGLEORDER $\wedge$ K-REALTIMEREADS($K$) $\wedge$ REALTIMEWW $\wedge$ SEQRVAL($\mathscr{F}$) |
| *7.- Fork-based models* | | |
| FORKLINEARISABLE($\mathscr{F}$) | [27] | REALTIME $\wedge$ NOJOIN $\wedge$ PRAM($\mathscr{F}$) |
| FORKSEQUENTIAL($\mathscr{F}$) | [28] | NOJOIN $\wedge$ PRAM($\mathscr{F}$) |
| FORK$^*$($\mathscr{F}$) | [29] | READYOURWRITES $\wedge$ REALTIME $\wedge$ RVAL($\mathscr{F}$) $\wedge$ ATMOSTONEJOIN |
| *8.- Per-object models* | | |
| SLOW($\mathscr{F}$) | [30] | PEROBJECTPRAM $\wedge$ RVAL($\mathscr{F}$) |
| CACHE($\mathscr{F}$) | [31] | PEROBJECTSINGLEORDER $\wedge$ SLOW($\mathscr{F}$) |
| PROCESSOR($\mathscr{F}$) | [31] | PEROBJECTSINGLEORDER $\wedge$ PRAM($\mathscr{F}$) |

This means that a CAP-free model is respected when the network is partitioned. On the other hand, a CAP-constrained model is not respected by every execution in case of network partitions:

**Definition 3.2** (CAP-constrained consistency model)**.** *CM is CAP-constrained if there is an execution $E$ in $\mathscr{E}_P$ that does not fulfil all consistency predicates that define CM.*

*Formally:* $\exists E \in \mathscr{E}_P : E \not\models \mathscr{P}_1 \wedge \cdots \wedge \mathscr{P}_n$.

Def. 3.2 cannot consider every execution in $\mathscr{E}_P$ since there may be executions in a partitioned system that still comply with all *CM* predicates. For instance, let consider execution $E_1 \in \mathscr{E}_P$ with $S = (\{p_1, p_2\}, \{x, y\})$ and a partition with $P_1 = \{p_1\}$, $P_2 = \{p_2\}$, $it = 4$ and $et = 10$: $E_1 = (\{o_1 = (p_1, wr, x, 2, \sqcup, 0, 1), o_2 = (p_2, rd, x, \sqcup, 2, 2, 3), o_3 = (p_2, wr, x, 3, \sqcup, 3, 4), o_4 = (p_1, wr, y, \sqcup, 2, 4, 5), o_5 = (p_2, wr, x, 6, \sqcup, 5, 6), o_6 = (p_1, rd, y, \sqcup, 2, 7, 8), o_7 = (p_1, rd, x, \sqcup, 6, 11, 12), o_8 = (p_2, rd, y, \sqcup, 2, 13, 14)\}, \{(o_1, o_2), (o_2, o_3), (o_3, o_4), (o_4, o_5), (o_5, o_6), (o_6, o_7), (o_7, o_8)\}, E_1.vis)$.

Processes $p_1$ and $p_2$ have remained active in the partition interval $(4, 10)$. Prop. 2.1 is respected. $E_1$ complies with all predicates that define the LINEARISABLE($\mathscr{F}$) model. As it will be stated in Theorem 4.1, LINEARISABLE($\mathscr{F}$) is CAP-constrained. This shows that some executions of CAP-constrained models in partitioned systems still respect the predicates that define their consistency model.

In order to compare consistency models in $S$, let use $\mathscr{E}^{CM}$ as the set of executions admitted in $S$ by a consistency model *CM*. This allows the following definition:

**Definition 3.3** (*Weaker than* ($\rightarrow$) relation on models)**.** *A model $A \equiv \mathscr{P}_{A,1} \wedge \cdots \wedge \mathscr{P}_{A,n}$ is weaker than another model $B \equiv \mathscr{P}_{B,1} \wedge \cdots \wedge \mathscr{P}_{B,m}$, i.e., $A \rightarrow B$, when the set of executions admitted by B is a proper subset of the set of executions admitted by A.*

*Formally:* $A \rightarrow B \equiv \mathscr{E}^B \subset \mathscr{E}^A$.

*Or:* $A \rightarrow B \equiv \mathscr{P}_{B,1} \wedge \cdots \wedge \mathscr{P}_{B,m} \Rightarrow \mathscr{P}_{A,1} \wedge \cdots \wedge \mathscr{P}_{A,n}$.

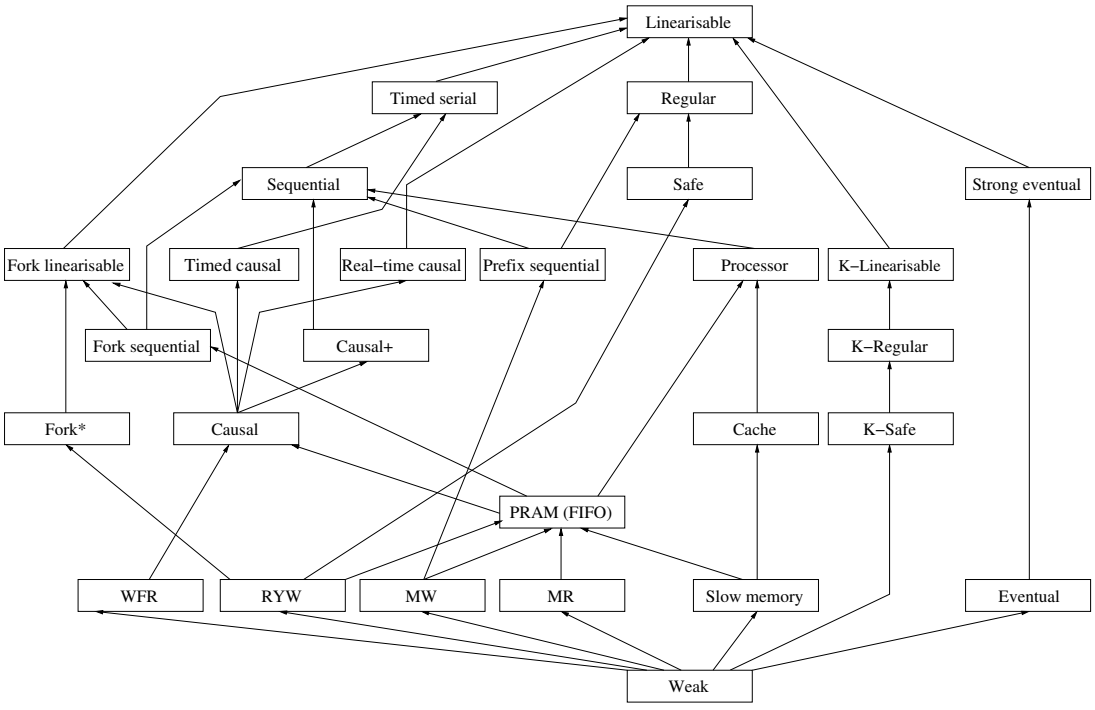


Figure 1: Strength-based partial ordering of consistency models.

When $A \rightarrow B$, model $B$ is stronger than $A$. Besides, when $A \not\rightarrow B \wedge B \not\rightarrow A$ then $A$ and $B$ are incomparable. Figure 1 depicts those relations. This initial model hierarchy is adapted from [12]. From Def. 3.3, combined with Def. 3.1 and 3.2, the following propositions can be stated.

**Proposition 3.1** (Freedom of weaker CAP-free models)**.** *Given two consistency models $A \equiv \mathscr{P}_{A,1} \wedge \cdots \wedge \mathscr{P}_{A,n}$ and $B \equiv \mathscr{P}_{B,1} \wedge \cdots \wedge \mathscr{P}_{B,m}$, if $A \rightarrow B$ and B is CAP-free, then A is also CAP-free.*

*Proof.* By Def. 3.3: (1) $\mathscr{P}_{B,1} \wedge \cdots \wedge \mathscr{P}_{B,m} \Rightarrow \mathscr{P}_{A,1} \wedge \cdots \wedge \mathscr{P}_{A,n}$. Additionally, since $B$ is CAP-free, according to Def. 3.1: (2) $\forall E \in \mathscr{E}_P^B : E \models \mathscr{P}_{B,1} \wedge \cdots \wedge \mathscr{P}_{B,m}$.

Let assume that $A$ was CAP-constrained. In that case, according to Def. 3.2: $\exists E_1 \in \mathscr{E}_P : E_1 \not\models \mathscr{P}_{A,1} \wedge \cdots \wedge \mathscr{P}_{A,n}$. This would imply that $\mathscr{P}_{A,1} \wedge \cdots \wedge \mathscr{P}_{A,n}$ is false in some executions when network partitions arise. If so, due to (1), $\mathscr{P}_{B,1} \wedge \cdots \wedge \mathscr{P}_{B,m}$ is neither generally true in case of partitions. So, an execution $E_2$ could be found such that $E_2 \in \mathscr{E}_P : E_2 \not\models \mathscr{P}_{B,1} \wedge \cdots \wedge \mathscr{P}_{B,m}$ and this implies that $B$ is CAP-constrained. This sets a contradiction with (2). Thus, $A$ is CAP-free. □

**Proposition 3.2** (Constriction of stronger CAP-constrained models)**.** *Given two consistency models $A \equiv \mathscr{P}_{A,1} \wedge \cdots \wedge \mathscr{P}_{A,n}$ and $B \equiv \mathscr{P}_{B,1} \wedge \cdots \wedge \mathscr{P}_{B,m}$, if $A \rightarrow B$ and $A$ is CAP-constrained, then $B$ is also CAP-constrained.*

*Proof.* By Def. 3.3: (1) $\mathscr{P}_{B,1} \wedge \cdots \wedge \mathscr{P}_{B,m} \Rightarrow \mathscr{P}_{A,1} \wedge \cdots \wedge \mathscr{P}_{A,n}$. Additionally, since $A$ is CAP-constrained, according to Def. 3.2: (2) $\exists E_1 \in \mathscr{E}_P : E_1 \not\models \mathscr{P}_{A,1} \wedge \cdots \wedge \mathscr{P}_{A,n}$.

Let assume that $B$ was CAP-free. In that case, according to Def. 3.1: $\forall E \in \mathscr{E}_P^B : E \models \mathscr{P}_{B,1} \wedge \cdots \wedge \mathscr{P}_{B,m}$. This would imply that $\mathscr{P}_{B,1} \wedge \cdots \wedge \mathscr{P}_{B,m}$ is true when network partitions arise. If so, due to (1), $\mathscr{P}_{A,1} \wedge \cdots \wedge \mathscr{P}_{A,n}$ is also true in case of partitions. Thus, $\forall E \in \mathscr{E}_P^A : E \models \mathscr{P}_{A,1} \wedge \cdots \wedge \mathscr{P}_{A,n}$ and therefore $A$ is CAP-free. This sets a contradiction with (2). So, $B$ is CAP-constrained. □

# 4 Finding a Consistency Border

Let us take the specifications of all consistency models as a base for analysing which requires any agreement that a network partition will break. Those models are *CAP-constrained*. Section 4.1 presents that analysis. Section 4.2 goes on in this analysis looking for other conditions that are not related with consensus and cannot be either attained in an available and partitioned system. Finally, Section 4.3 looks for convergence-based inter-model relations.

Most predicates in Table 1 strengthen the definition of the consistency models that include them. However, there are several exceptions: K-REALTIMEREADS(K), NOJOIN, ATMOSTONEJOIN and SEQRVAL($\mathscr{F}$). The first one breaks read determinism. Instead of returning the last received write, each process may read any of the K latest writes. The second and third ones impose divergence among processes once a failure happens. The last one is a weaker variant for RVAL($\mathscr{F}$), since it drops the determinism of read operations in case of concurrent writes. Because of this, all consistency models that include them should be analysed with care.

## 4.1 Starting Point: The Linearisable Model

Gilbert and Lynch [3] proved the CAP theorem assuming *linearisable* [15] consistency. Let us start revising its definition, given in Table 2, with the goal of identifying which predicates cannot be respected in a partitioned and available system:

LINEARISABLE($\mathscr{F}$) $\equiv$ SINGLEORDER $\wedge$ REALTIME $\wedge$ RVAL($\mathscr{F}$)

RVAL($\mathscr{F}$) defines the appropriateness of the return value in an operation based on its execution context. Its compliance is assumed in the following discussions. The other predicates mean the following (Table 1):

- REALTIME: $rb \subseteq ar$. This predicate states that all operations ordered by the returns-before ($rb$) relation are considered in the arbitration relation ($ar$). Since $rb$ considers real time, it is able to order the operations executed by different processes, even when they are placed in different network components in a partition.

  Relation $ar$ totally orders the operations in a history $H$. Considered in isolation, REALTIME may be respected in a partitioned system. However, it leads to contradictions when the *vis* relation assumed in other predicates requires communication among isolated network components.

  As a result, the inclusion of REALTIME in a consistency model may endanger the compliance with other *vis*-related predicates, e.g., SINGLEORDER.

- SINGLEORDER: $\exists H' \subseteq \{op \in H : op.oval = \nabla\} : vis = ar \setminus (H' \times H)$. This predicate states that the visibility relation (*vis*) coincides with the arbitration relation (*ar*), discarding incomplete operations. Let us prove that SINGLEORDER cannot be respected while a temporary network partition lasts in a system.

  **Lemma 4.1** (SINGLEORDER $\wedge$ RVAL($\mathscr{F}$) unattainability)**.** *In a system $S = (P,O)$ that uses a consistency model that includes* SINGLEORDER*, while a network partition NP arises, there is some execution $E \in \mathscr{E}_P$ in which* SINGLEORDER $\wedge$ RVAL*($\mathscr{F}$) is false.*

  *Proof.* Without loss of generality, let us assume the following two conditions:

  1. $\exists E \in \mathscr{E}_P$: $E \models$ SINGLEORDER $\wedge$ RVAL($\mathscr{F}$).
  2. Partition $NP$ has happened in the interval $(it, et)$ generating two different separate network components $P_1$ and $P_2$; i.e., $NP = (S, \{1,2\}, it, et)$. $\exists p_1, p_2 \in P : p_1 \in P_1, p_2 \in P_2$.

  Process $p_1$ has executed $op_{1,1} = (p_1, wr, x, v, \sqcup, st, st+1)$. Besides, $p_2$ has also run $op_{2,1} = (p_2, wr, x, v', \sqcup, st', st'+1)$ with both $st, st' \in (it, et-1)$. Process $p_1$ also runs $op_{1,2} = (p_1, rd, x, \sqcup, v, st'', st''+1)$ with $st'' \in (it, et-1)$ and $st < st' < st''$.

  Since $op_{1,1}$ and $op_{1,2}$ have happened both in $p_1$, they are trivially related by *vis*: $(op_{1,1}, op_{1,2}) \in vis$. This justifies the read value in $op_{1,2}$ according to RVAL($\mathscr{F}$).

  By definition, *ar* orders all operations in $E$. Since RVAL($\mathscr{F}$) is respected in $E$, the order set by *ar* should allow that $op_{1,2}$ reads value $v$. This may only happen when $E.ar$ is either $ar_1 = \{(op_{2,1}, op_{1,1}), (op_{1,1}, op_{1,2})\}$ or $ar_2 = \{(op_{1,1}, op_{1,2}), (op_{1,2}, op_{2,1})\}$. Let us assume that $E.ar = ar_1$. Condition (1) above states that SINGLEORDER is true in $E$. Thus, $E.vis = E.ar$.

  Condition (2), above, states that $S$ remained partitioned while $E$ was run. Because of condition 4 from Def. 2.1, $p_1$ and $p_2$ cannot exchange messages in $(it, et)$. This means that $(op_{1,*}, op_{2,1}) \notin vis \wedge (op_{2,1}, op_{1,*}) \notin vis$; i.e., $p_1$ and $p_2$ operations cannot be directly or transitively related in *vis*. So, $E.vis \neq E.ar$. Therefore, this sets a contradiction with our assumed condition (1); i.e., execution $E$ cannot respect SINGLEORDER $\wedge$ RVAL($\mathscr{F}$). □

  Lemma 4.1 assumes a single object $x$ in its proof. So, this result applies to other predicates that constrain the $vis = ar$ equality to each object considered in isolation, e.g., to the PEROBJECTSINGLEORDER predicate.

These predicates have been used in the definition of several consistency models. Models based on SINGLEORDER or PEROBJECTSINGLEORDER will be CAP-constrained when no relaxing predicate is used in their definition. On the other hand, those using SINGLEORDER and any relaxing predicate, and those based on REALTIME should be further assessed. Let us revise which they are:

- SINGLEORDER with no relaxing predicate: *linearisable*, *regular*, *sequential*, *processor*, *cache* and *timed serial*.

  Let us refer to these models as *strong*-SINGLEORDER models. *SSO* is the set of those models. All they are CAP-constrained, as stated in the following theorem.

  **Theorem 4.1** (Strong-SINGLEORDER models are CAP-constrained)**.** *The* LINEARISABLE*($\mathscr{F}$),* REGULAR*($\mathscr{F}$),* SEQUENTIAL*($\mathscr{F}$),* PROCESSOR*($\mathscr{F}$),* CACHE*($\mathscr{F}$) and* TIMEDSERIAL*($\mathscr{F}, \Delta$) models are CAP-constrained.*

  *Proof.* All models in *SSO* include the SINGLEORDER $\wedge$ RVAL($\mathscr{F}$) conjunction. They do not include any other predicate that relaxes what is required in that conjunction. Formally: $\forall M_j \in SSO, M_j \equiv \mathscr{P}_{j,1} \wedge \cdots \wedge \mathscr{P}_{j,n_j}$. $\exists i, 1 \leq i \leq n_j : \mathscr{P}_{j,i} =$ SINGLEORDER $\wedge$ RVAL($\mathscr{F}$).

  Let us assume a system $S$ with model $M_j$. Then, predicate $\mathscr{P}_{j,i}$ allows the application of Lemma 4.1. As a result: $\exists E \in \mathscr{E}_P, E \not\models \mathscr{P}_{j,1} \wedge \cdots \wedge \mathscr{P}_{j,n_j}$, since $\mathscr{P}_{j,i}$ may be false in those executions. Thus, by Def. 3.2, each $M_j$ is CAP-constrained. □

- SINGLEORDER with relaxing predicates: *safe*, *k-linearisable*, *k-regular* and *k-safe*.

  Let us start our analysis with the *safe* model. It relaxes SINGLEORDER with the conjunction of the SEQRVAL($\mathscr{F}$) predicate. The latter only requires determinism in a read operation $op_r$ when there are no other concurrent write operations with $op_r$. When there are concurrent write operations, $op_r$ may return any value. Considering that behaviour, let us prove that the SAFE($\mathscr{F}$) model is CAP-constrained.

  **Theorem 4.2** (SAFE($\mathscr{F}$) is CAP-constrained)**.** *The* SAFE*($\mathscr{F}$) consistency model is CAP-constrained.*

  *Proof.* Let us assume an execution $E_s \in \mathscr{E}_P$ with model SAFE($\mathscr{F}$). Let us consider that there are two network components $P_1$ and $P_2$ in $E_s$, each one consisting of a single process, $p_0$ and $p_1$, respectively. Let us imagine that in $E_s$ there are no read events in process $p_i$ concurrent with write events in process $p_{1-i}$. This may easily happen if the set of objects being managed by processes $p_0$ and $p_1$ is large, since each time one process is writing on a given object $o_k$ the other process may be reading other objects. In this scenario, SEQRVAL($\mathscr{F}$) is equivalent to RVAL($\mathscr{F}$). As a result, $E_s \not\models$SAFE($\mathscr{F}$), since Lemma 4.1 may then be applied onto $E_s$. Because of this, considering Def. 3.2, SAFE($\mathscr{F}$) is CAP-constrained. □

  The remaining models (*k-linearisable*, *k-regular* and *k-safe*) share K-REALTIMEREADS(K) as their relaxing predicate. Let us refer to them as *k-staleness* models. Like SEQRVAL($\mathscr{F}$), K-REALTIMEREADS(K) relaxes read determinism. In this case, each read operation may return any of the $K$ latest written values in $S$. Let us consider this predicate in the following lemma.

  **Lemma 4.2** (K-REALTIMEREADS(K) $\wedge$ SINGLEORDER unattainability)**.** *In a system $S = (P, O)$ that uses a consistency model with* K-REALTIMEREADS(K) $\wedge$ SINGLEORDER*, while a network partition NP arises in a time interval (it,et), there is some execution $E \in \mathscr{E}_P$ in which* K-REALTIMEREADS(K) $\wedge$ SINGLEORDER *is false.*

  *Proof.* Let us assume that there are at least two network components in $S$, being $P_1$ and $P_2$ two of those components, with at least one process in each component, $p_1$ and $p_2$, respectively. $E_{rt}$ is an execution in $\mathscr{E}_P$. In $E_{rt}$, process $p_1$ has written $K$ different values on object $x$ after time $it$, in operations $op_{1,1} \ldots op_{1,K}$, respectively. Later, $p_2$ reads $x$ in operation $op_{2,1}$. Process $p_2$ had not written any value on $x$ since time $it$. All those operations have happened before time $et$.

  According to K-REALTIMEREADS(K), several of these $K$ latest writes from $p_1$ precede $op_{2,1}$ in the $ar$ relation. Thus, conceptually, $p_2$ reads any of those values in $op_{2,1}$. However, SINGLEORDER requires that all completed operations in $ar$ are also in $vis$. This means that the effects from several $op_{1,1} \ldots op_{1,K}$ must be already delivered to $p_2$ when $op_{2,1}$ is started. Since $p_1$ and $p_2$ remain disconnected in the $(it, et)$ interval, according to condition 4 from Def. 2.1, the values written by $p_1$ are not visible to $op_{2,1}$. So, $vis \neq ar$ in $E_{rt}$, i.e., SINGLEORDER is not true in $E_{rt}$.

  Therefore, there are executions that do not comply with K-REALTIMEREADS(K) $\wedge$ SINGLEORDER in $S$. □

  With that result, the following theorem may be proved.

  **Theorem 4.3** (K-staleness models are CAP-constrained)**.** K-REGULAR*($\mathscr{F}$,K),* K-SAFE*($\mathscr{F}$,K) and* K-LINEARISABLE*($\mathscr{F}$,K) are CAP-constrained.*

  *Proof.* Immediate from Lemma 4.2 and Def. 3.2. □

- REALTIME (or its variants REALTIMEWRITES, REALTIMEWW): *linearisable*, *regular*, *safe*, *k-linearisable*, *fork linearisable*, and *fork*$^*$. From these models, *fork linearisable* and *fork*$^*$ have not been considered yet in the analysis, since the others include SINGLEORDER in their predicates. Besides, *fork linearisable* and *fork*$^*$ contain in their definitions, respectively, the NOJOIN and ATMOSTONEJOIN relaxing predicates that have not been assessed yet.

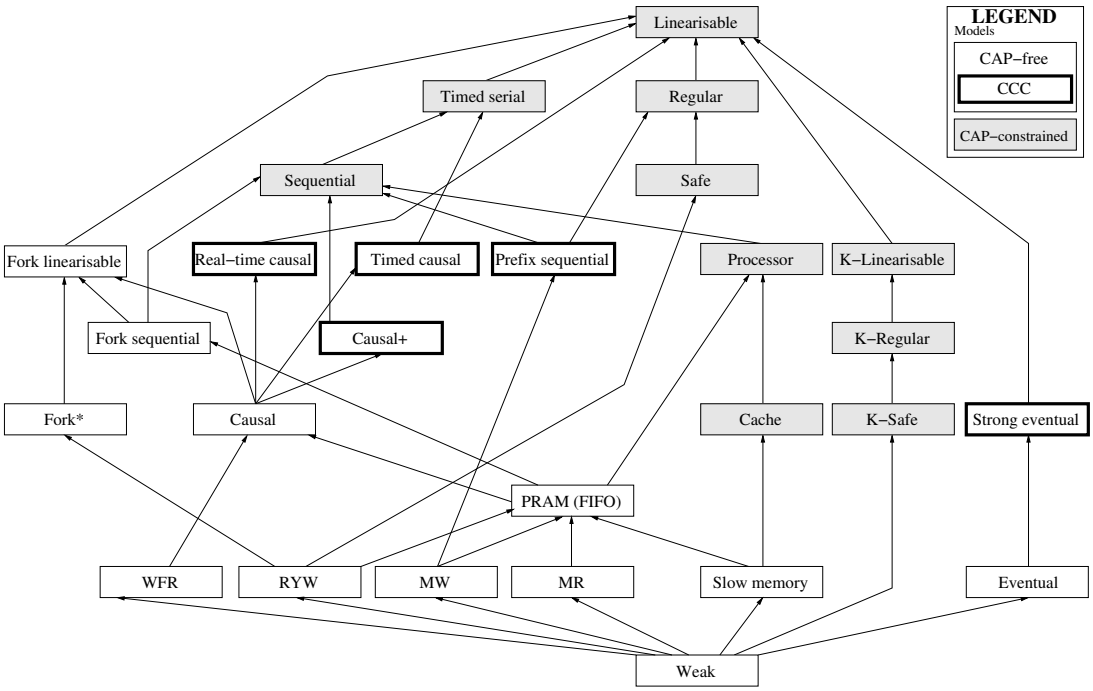


Figure 2: CAP-constrained and CAP-free consistency models considering SINGLEORDER and REAL-TIME.

It is worth noting that fork-based models were proposed by Mazières and Shasha in [27] with the aim of dealing with Byzantine failures in the management of networked file systems. When a failure of that kind arises, some users may create a new version of a given file without considering the latest version kept in the servers. This generates a new branch (i.e., a fork) of versions thereafter. If multiple failures exist, the resulting set of versions defines a tree of divergent branches and those branches cannot join again. The specification shown in Tables 1 and 2 is general and does not consider specific failure models. Our discussion considers those generalised consistencies. In them, a fork in the version tree might be caused by a network partition. That possibility was not explicitly assumed in the original fork-based papers [27, 28, 29].

Let us assess both models.

**Theorem 4.4** (FORKLINEARISABLE($\mathscr{F}$) is CAP-free)**.** *The* FORKLINEARISABLE*($\mathscr{F}$) consistency model is CAP-free.*

*Proof.* Let us assume a system $S$ where all its executions in $\mathscr{E}^C$ respect PRAM $\wedge$ REALTIME $\wedge$ NOJOIN $\wedge$ RVAL($\mathscr{F}$). Let us imagine that a network partition $NP = (S, K, it, et)$ arises in $S$. Let analyse what happens with each one of those predicates in the $(it, et)$ interval:

1. REALTIME is trivially true, since it imposes a method for building *ar* based on the *rb* relation. That method is implemented by a replication protocol and it can be maintained in spite of *NP*. Actually, it is only a criterion for ordering invisible conflicting writes. That criterion may still be used in case of network partitions.
2. RVAL($\mathscr{F}$) should be also true. This predicate states which criteria should be followed for deciding the return value of each operation. Again, it is embedded in the replication protocol and does not change when a network partition happens.
3. NOJOIN states that when two different sessions (i.e., processes) define a pair of operations with an operation per process that are related by *ar* but are not related by *vis*, then all subsequent operations in each of those sessions will not visible for the other process.

In case of a network partition, if $p_i \in P_i$, $p_j \in P_j$ and $1 < i, j <\mid K \mid$, then all operations run by $p_i$ and $p_j$ will be ordered by $ar$ according to the $rb$ relation, but there will not be any pair $(a,b) \in vis$ with $a.proc = p_i$ and $b.proc = p_j$ (or $a.proc = p_j$ and $b.proc = p_i$) while the partition lasts, because of condition 4 from Def. 2.1.

Inter-process connectivity is recovered after time $et$. However, this will not resume inter-process visibility, since the replication protocol being used in $S$ respects NOJOIN while no partitions arise. This means that none of the subsequent operations run by $p_i$ and $p_j$ will be visible to the other process. This ensures that NOJOIN remains true in all cases: while partitions exist and while connectivity is healthy.

4. PRAM demands that $so$ is respected in $vis$. Since $so$ is an order, this basically means that $vis$ cannot contain any pair of operations that contradicts any of the pairs held in $so$. In order to guarantee PRAM, the replication protocol in $S$ must transfer the writes made in each session in FIFO order to the remaining processes. That transfer order is not endangered nor prevented by network partitions. Indeed, when no write is transferred to the remaining processes, PRAM is still true, since no $so$ contradiction may arise in $vis$. Therefore, PRAM is respected in $\mathscr{E}^P$.

Since all the predicates are true in case of network partitions, $\forall E \in \mathscr{E}^P : E \models$ FORKLINEARISABLE($\mathscr{F}$). Therefore, according to Def. 3.1, FORKLINEARISABLE($\mathscr{F}$) is CAP-free. □

**Corollary 4.1** (FORK$^*$($\mathscr{F}$) is CAP-free)**.** *The* FORK$^*$*($\mathscr{F}$) consistency model is CAP-free.*

*Proof.* FORK$^*$($\mathscr{F}$) is weaker than FORKLINEARISABLE($\mathscr{F}$) and, according to Theorem 4.4, the latter is CAP-free. Thus, due to Proposition 3.1, FORK$^*$($\mathscr{F}$) is also CAP-free. □

After this analysis on SINGLEORDER and REALTIME, a preliminary borderline between CAP-constrained and CAP-free models may be set as depicted in Figure 2.

## 4.2 Exploring the Frontier

Considering Figure 2, let us continue our analysis revising which are the predicates that define the set *CCC* (*CAP-constrained candidates*) of strongest models that do not need the SINGLEORDER or REALTIME conditions in their specifications. In this scope, "strongest" refers to those models that: (1) are not tagged yet as CAP-constrained, (2) are directly related as "weaker than" a CAP-constrained model, and (3) do not have any other CAP-free model stronger than them. According to Figure 2, these *CCC* models are: *strong eventual*, *causal+*, *timed causal*, *real-time causal* and *prefix sequential*. The *fork linearisable* model also complies with those three conditions, but Theorem 4.4 has already proved that it is CAP-free.

If any of these models was also CAP-constrained, we would continue our analysis with the new candidates generated by its inclusion in the CAP-constrained set. On the other hand, if no *CCC* model is identified as CAP-constrained in this stage, other models more relaxed than those in *CCC* will be also CAP-free, according to Prop. 3.1. In that case, our analysis should end there.

Part of these *CCC* models have been proved CAP-free either in the papers that proposed them or in other recent works. That is the case of STRONGEVENTUAL($\mathscr{F}$) [17], CAUSAL+($\mathscr{F}$) [22], REALTIMECAUSAL($\mathscr{F}$) [9] and PREFIXSEQUENTIAL($\mathscr{F}$) [32]. Therefore, they are CAP-free by definition.

Let us consider now the other model: *timed causal*.

**Theorem 4.5.** *The* TIMEDCAUSAL*($\mathscr{F},\Delta$) consistency model is CAP-constrained.*

*Proof.* Let us assume an execution $E \in \mathscr{E}_C : E \models$ TIMEDCAUSAL($\mathscr{F},\Delta$). Let a partition $NP$ occur in $S$ with $NP.et - NP.it > \Delta$. Let $E' \in \mathscr{E}_P : E'.H$ is an adaptation of $E.H$. There may be $(a,b)$ pairs (with $a.type = wr$ and $b.type = rd$) in $E'.vis$ generated at network reconnection time with $a.rt = NP.it$ and $b.st > NP.et$. Note that $b$ needs to receive the value written in $a$, and that transmission may only complete after $NP.et$. Since $NP.et - NP.it > \Delta$, then $b.st > a.rt + \Delta$. Thus, $E'$ makes TIMEDVISIBILITY($\Delta$) false and this means that $E' \not\models$ TIMEDCAUSAL($\mathscr{F},\Delta$). Therefore, by Def. 3.2, TIMEDCAUSAL($\mathscr{F},\Delta$) is CAP-constrained. □

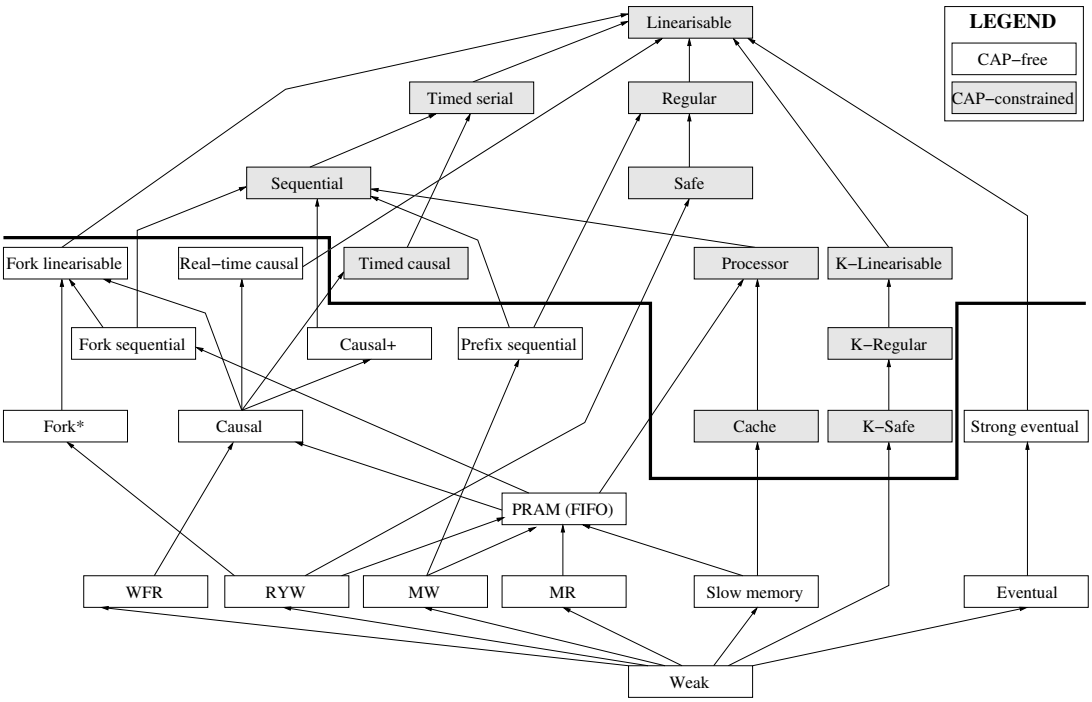


Figure 3: CAP-constrained vs CAP-free borderline.

This analysis has shown that the *timed causal* model is CAP-constrained. Figure 2 shows that *timed causal* is directly stronger than *causal* and there is no other weaker model directly related with *timed causal*. The *causal* model is weaker than *causal+* and the latter is CAP-free. Thus, according to Prop. 3.1, CAUSAL($\mathscr{F}$) is CAP-free. This means that no other CAP-constrained model may be found. Therefore, the borderline between CAP-constrained and CAP-free models is finally set as depicted in Figure 3.

## 4.3 Revising Inter-Model Relationships

Previous figures show some "weaker than" relations between models. However, there are some other inter-model relations that have not been depicted yet. Those relations are explained hereafter. Section 4.3.1 discusses configurable models. Later, Section 4.3.2 describes relations that deal with state convergence.

### 4.3.1 Configurable Models

The family of *probabilistic bounded staleness* (PBS) [33] models consists of three models: PBS k-staleness, PBS t-visibility and PBS (k,t)-staleness. They are intended for quorum-based eventually consistent datastores. The first describes a probabilistic model that restricts the staleness of read values. The second limits probabilistically the time needed by written values to become visible. The third one combines the other two. PBS k-staleness, depending on its parameter $k$, may be weaker than or equivalent to *k-linearisable*, while PBS t-visibility is a probabilistic weakening of TIMEDVISIBILITY($\Delta$). Their place around the CAP-constrained vs CAP-free frontier depends on the values of their parameters, and this avoids their inclusion in Figure 4. They are generally configured as eventually consistent models, and this implies CAP-free models. However, PBS k-staleness when configured as non-stale becomes k-linearisable and, as such, is CAP-constrained.

As Brewer states in [8] "because partitions are rare, there is little reason to forfeit C (consistency) or A (availability) when the system is not partitioned". This means that a strong model is needed while the network shows no connectivity problem and such consistency should be only relaxed when a network

partition arises. Dynamically configurable models that may relax their consistency, as those proposed in the PBS family, seem to be an adequate solution to this problem. There have been several other models (e.g., [34, 35, 36, 37, 38, 39, 40]) of this kind, that are analysed by Viotti and Vukolić [12] in their eighth group: *composable and tunable* models. We refer the reader to [12] for a short description and comparison of them. Many of those models admit configurations in both parts of our identified frontier.

### 4.3.2 Convergence-based Relations

*Eventual* consistency was defined with the goal of breaking the constraints of the CAP theorem [11]. Eventual models that reach state convergence once their processes have seen the same set of write operations, like StrongEventual($\mathscr{F}$) [17], may be weaker than most CAP-constrained models. Indeed, according to Prop. 3.1, StrongEventual($\mathscr{F}$) cannot be stronger than any CAP-constrained model, since it is CAP-free [17]. Let us proceed in this analysis comparing several families of protocols:

- Cache($\mathscr{F}$) and stronger models.

  If StrongEventual($\mathscr{F}$) was weaker than Cache($\mathscr{F}$), then Cache($\mathscr{F}$) predicates would imply those that define StrongEventual($\mathscr{F}$). Let us see whether that is true. Several lemmas are needed to this end.

  **Lemma 4.3.** PerObjectSingleOrder $\wedge$ RVal($\mathscr{F}$) $\Rightarrow$ EventualVisibility $\wedge$ RVal($\mathscr{F}$).

  *Proof.* Without loss of generality, let assume a system $S = (\{p_1, p_2\}, \{x\})$ where all its executions should respect PerObjectSingleOrder. Let consider executions in which all operations end. PerObjectSingleOrder implies: (1) $\forall E \in \mathscr{E}, E.vis \cap ob = E.ar \cap ob$.

  Let us assume that EventualVisibility did not hold in $S$. This means that: (2) $\forall a \in E.H, \forall [f] \in E.H/\approx_{ss}: b \in [f] \wedge (a,b) \in rb \Rightarrow (a,b) \notin E.vis$, i.e., all $rb$-ordered operations that do not belong to the same process are not in $vis$. In that scenario, execution $E_3$ would match $\neg$EventualVisibility in $S$: $E_3 = (\{o_1 = (p_1, wr, x, 3, \sqcup, 1, 2), o_2 = (p_2, wr, x, 2, \sqcup, 1, 2), o_3 = (p_1, rd, x, \sqcup, 3, 3, 4), o_4 = (p_2, rd, x, \sqcup, 2, 4, 5), o_5 = (p_1, wr, x, 5, \sqcup, 6, 7), o_6 = (p_1, rd, x, \sqcup, 5, 8, 9)\}, \{(o_1, o_3), (o_2, o_4), (o_3, o_5), (o_5, o_6)\}, \{(o_1, o_3), (o_3, o_5), (o_5, o_6), (o_6, o_2), (o_2, o_4)\})$. But now, $E_3$ violates condition (1), since $E_3.vis \neq E_3.ar$. Thus, a contradiction is reached. So, EventualVisibility must hold in $S$. Therefore: PerObjectSingleOrder $\wedge$ RVal($\mathscr{F}$) $\Rightarrow$ EventualVisibility $\wedge$ RVal($\mathscr{F}$). □

  **Lemma 4.4.** NoCircularCausality *is true in message-based inter-process communication systems.*

  *Proof.* By definition, $hb \equiv (so \cup vis)^+$, $vis$ is acyclic, $so \equiv rb \cap ss$ and $rb \equiv \{(a,b) : a,b \in H \wedge a.rt < b.st\}$. Because of $rb$, $so$ is also acyclic, since the operations that compose the $rb$ elements are ordered by real time. Thus, to close a cycle, $vis$ needs elements that transitively relate two write operations in a reverse real-time order. This contradicts the $vis$ definition, since $vis$ represents value propagation among processes. Section 2 has stated that such value propagation is message-based. Therefore, when two operations are related by $vis$, their relation order is consistent with real time. As a result, $hb$ is acyclic and NoCircularCausality is true. □

  According to Burckhardt, NoCircularCausality was introduced in his specification of *basic eventual* consistency in order to avoid the *circular causality* [14] (also known as *thin air* [13]) anomaly that may be generated by some compiler optimisations for shared memory multiprocessors (since $vis$ is not acyclic in that context). However, in scenarios where those optimisations cannot be applied, as it is assumed in our paper, such an anomaly cannot happen.

  **Lemma 4.5.** PerObjectSingleOrder $\wedge$ RVal($\mathscr{F}$) $\Rightarrow$ StrongConvergence $\wedge$ RVal($\mathscr{F}$).

*Proof.* STRONGCONVERGENCE requires that when two read operations ($\forall a,b \in H\ |_{rd}$, on two different processes) have been preceded by the same set of write operations ($vis^{-1}(a)\ |_{wr} = vis^{-1}(b)\ |_{wr}$), then both reads return the same value ($a.oval = b.oval$). PEROBJECTSINGLEORDER implies that every process in $S$ sees the same sequence of write operations on each object. Therefore, every pair of processes that respect PEROBJECTSINGLEORDER have seen the same sequence of values. In that case, when two different processes compare the output value of a read in each process preceded by the same set of write operations, RVAL($\mathscr{F}$) guarantees that they read the same value. Thus, PEROBJECTSINGLEORDER $\wedge$ RVAL($\mathscr{F}$) $\Rightarrow$ STRONGCONVERGENCE $\wedge$ RVAL($\mathscr{F}$). □

**Theorem 4.6.** STRONGEVENTUAL*($\mathscr{F}$)* $\rightarrow$ CACHE*($\mathscr{F}$).*

*Proof.* Lemmas 4.3, 4.4 and 4.5 imply that: PEROBJECTSINGLEORDER $\wedge$ RVAL($\mathscr{F}$) $\Rightarrow$ STRONGCONVERGENCE $\wedge$ EVENTUALVISIBILITY $\wedge$ NOCIRCULARCAUSALITY $\wedge$ RVAL($\mathscr{F}$). PEROBJECTPRAM constrains how written values are propagated. So, it is not a relaxing predicate in this scope. Therefore, according to Def. 3.3, STRONGEVENTUAL($\mathscr{F}$) $\rightarrow$ CACHE($\mathscr{F}$). □

**Corollary 4.2.** STRONGEVENTUAL*($\mathscr{F}$) is weaker than* PROCESSOR*($\mathscr{F}$),* SEQUENTIAL*($\mathscr{F}$),* TIMEDSERIAL*($\mathscr{F}$) and* LINEARISABLE*($\mathscr{F}$).*

*Proof.* All mentioned models are stronger than CACHE($\mathscr{F}$). Relation *weaker than* (Def. 3.3) is transitive. So, because of Theorem 4.6, STRONGEVENTUAL($\mathscr{F}$) is weaker than each one of them. □

- SAFE($\mathscr{F}$).

  In the SAFE($\mathscr{F}$) model, a read operation $o_r$ returns the value written in the latest write if there is no write concurrent with $o_r$. If any concurrent write exists, $o_r.oval$ may be any value. This provides the base for the following theorem.

  **Theorem 4.7.** STRONGEVENTUAL*($\mathscr{F}$)* $\not\rightarrow$ SAFE*($\mathscr{F}$).*

  *Proof.* Let us assume a system $S = (\{p_1, p_2, p_3\}, \{x\})$. In order to prove that a model $A$ is not weaker than another model $B$, we should find an execution $E_{nw}$ that complies with $B$ and does not respect $A$, since this implies that $\mathscr{E}^B \not\subset \mathscr{E}^A$. So, $E_4 = (\{o_1 = (p_1, wr, x, 2, \sqcup, 0, 1),\ o_2 = (p_1, wr, x, 1, \sqcup, 2, 5), o_3 = (p_2, rd, x, \sqcup, 15, 3, 6), o_4 = (p_3, rd, x, \sqcup, 20, 4, 7)\}, \{(o_1, o_2), (o_2, o_3), (o_3, o_4)\}, \{(o_1, o_2),\ (o_2, o_3),\ (o_3,\ o_4)\})$ satisfies those requirements. In that execution, $p_1$ has written values 2 and 1 on $x$, in that order. That means that value 2 was known (i.e., visible) to $p_1$ when it wrote 1. The replication protocol forwarded value 1 to process $p_2$ while it was written, and $p_2$ also propagated that value 1 to $p_3$. These actions explain the three pairs in the $E_4.vis$ relation.

  $E_4$ complies with all predicates that define SAFE($\mathscr{F}$):

  1. SINGLEORDER needs that $E_4.vis = E_4.ar$, and that is true,
  2. REALTIMEWRITES ($rb\ |_{wr \rightarrow op} \subseteq ar$) means that, in $E_4$, $(o_1, o_2)$, $(o_1, o_3)$ and $(o_1, o_4)$ must be in $ar$. Transitively, those pairs of operations are in $E_4.ar$.
  3. SEQRVAL($\mathscr{F}$) allows that both $o_3$ and $o_4$ return any value as a result of the read operation, since $o_2$ is a write concurrent with both of them.

  However, $E_4$ does not respect STRONGCONVERGENCE, since $E_4.vis^{-1}(o_3)\ |_{wr} = E_4.vis^{-1}(o_4)\ |_{wr} = \{o_1, o_2\}$, but $o_3.oval \neq o_4.oval$.

  Therefore, $\mathscr{E}^{Safe} \not\subset \mathscr{E}^{StrongEventual}$. Then, according to Def. 3.3, STRONGEVENTUAL($\mathscr{F}$) $\not\rightarrow$ SAFE($\mathscr{F}$). □

Let us find out which models are weaker than SAFE($\mathscr{F}$) in order to adequately show as many weaker-than relations as possible in Figure 4. To this end, we need to prove a property that was previously conjectured:

**Proposition 4.1.** K-REALTIMEREADS(K) *is a weakening predicate.*

*Proof.* According to its definition, $\forall a \in H \mid_{wr}, \forall b \in H \mid_{rd}, \forall PW \subseteq H \mid_{wr}, \forall pw \in PW :\mid PW \mid < K \wedge (a,pw) \in ar \wedge (pw,b) \in rb \wedge (a,b) \in rb \Rightarrow (a,b) \in ar$, when $b$ is a read operation, it may get as its result any of the latest $K$ written values.

Let us consider a consistency model $M$ that is built as the conjunction of a set of consistency predicates: $M \equiv \mathscr{P}_1 \wedge \cdots \wedge \mathscr{P}_n$. We may build another consistency model $M'$ with this definition: $M' \equiv \mathscr{P}_1 \wedge \cdots \wedge \mathscr{P}_n \wedge$ K-REALTIMEREADS(K).

Let us assume a generic execution $E \in \mathscr{E}^M$ that consists of at least $K$ write operations ($o_{w1} \ldots o_{wK}$, placed in that order in $E.ar$) and with $o_r$ as its first read operation after the latest of those $K$ writes. According to the RVAL($\mathscr{F}$) or SEQRVAL($\mathscr{F}$) contained in $M$, $o_r.oval = val$.

K-REALTIMEREADS(K) introduces indeterminism in the output values of read operations. However, according to the definition of K-REALTIMEREADS(K), $E$ still complies with all predicates that define $M'$. Therefore, $\mathscr{E}^M \subseteq \mathscr{E}^{M'}$. This implies that $E \in \mathscr{E}^{M'}$.

Let us check now whether $\mathscr{E}^M$ is a strict subset of $\mathscr{E}^{M'}$ or both sets are equal. To this end, let us replay $E$ using $M'$ as the consistency model in $S$. Each one of those replays may generate a result different to $val$ for $o_r.oval$. Now $o_r.oval$ is a value $val'$, with $val'$ taken from any of the $o_{wx}.ival$ (with $1 \leq x \leq K$) values. All those replays of $E$ with $val' \neq val$ will belong to $\mathscr{E}^{M'}$, but will not be acceptable in $\mathscr{E}^M$. Therefore, this shows that $\mathscr{E}^M \subset \mathscr{E}^{M'}$ and, according to Def. 3.3, this proves that $M' \rightarrow M$. □

Then, the following corollary may be stated.

**Corollary 4.3.** K-SAFE($\mathscr{F}$,K) $\rightarrow$ SAFE($\mathscr{F}$)

K-REGULAR($\mathscr{F}$,K) $\rightarrow$ REGULAR($\mathscr{F}$)

K-LINEARISABLE($\mathscr{F}$,K) $\rightarrow$ LINEARISABLE($\mathscr{F}$)

*Proof.* Every statement in this corollary is a direct consequence of Prop. 4.1. □

- K-REALTIMEREADS(K)-based models.

  The following proposition is needed for proving that all models in this set are incomparable to STRONGEVENTUAL($\mathscr{F}$).

**Proposition 4.2.** K-REALTIMEREADS(K) $\not\Rightarrow$ STRONGCONVERGENCE.

*Proof.* According to the K-REALTIMEREADS(K) definition, $\forall a \in H \mid_{wr}, \forall b \in H \mid_{rd}, \forall PW \subseteq H \mid_{wr}, \forall pw \in PW :\mid PW \mid < K \wedge (a,pw) \in ar \wedge (pw,b) \in rb \wedge (a,b) \in rb \Rightarrow (a,b) \in ar$, when $b$ is a read operation, it may get as its result any of the latest $K$ written values.

Let $S$ be a system where no communication problem exists. Let $E$ be an execution that consists of $K$ write operations ($o_{w1} \ldots o_{wK}$, placed in that order in $E.ar$) each one writing a different value on object $x$ and with $o_{r1}$ and $o_{r2}$ as two read operations after the latest of those $K$ writes. Since there are no communication problems, $vis^{-1}(o_{r1}) \mid_{wr} = \{o_{w1} \ldots o_{wK}\}$ and $vis^{-1}(o_{r2}) \mid_{wr} = vis^{-1}(o_{r1}) \mid_{wr}$. However, $o_{r1}.oval \neq o_{r2}.oval$, since $o_{r1}.oval$ may be $o_{w1}.ival$ and $o_{r2}.oval$ be $o_{wK}.ival$ and those values are different. By definition of STRONGCONVERGENCE, STRONGCONVERGENCE becomes false in $E$. □

**Corollary 4.4.** STRONGEVENTUAL($\mathscr{F}$) $\not\rightarrow$ K-SAFE($\mathscr{F}$,K).

*Proof.* By Prop. 4.2, $\exists E \in \mathscr{E}, E \models$ K-SAFE($\mathscr{F}$,K) $\wedge\ E \not\models$ STRONGEVENTUAL ($\mathscr{F}$). Thus, by Def. 3.3, STRONGEVENTUAL($\mathscr{F}$) $\not\rightarrow$ K-SAFE($\mathscr{F}$,K). □

**Corollary 4.5.** STRONGEVENTUAL($\mathscr{F}$) $\not\rightarrow$ K-REGULAR($\mathscr{F}$,K).

*Proof.* By Prop. 4.2, $\exists E \in \mathscr{E}, E \models$ K-REGULAR($\mathscr{F}$,K) $\wedge\, E \not\models$ STRONGEVENTUAL ($\mathscr{F}$). Thus, by Def. 3.3, STRONGEVENTUAL($\mathscr{F}$) $\not\rightarrow$ K-REGULAR($\mathscr{F}$,K). □

**Corollary 4.6.** STRONGEVENTUAL*($\mathscr{F}$)* $\not\rightarrow$ K-LINEARISABLE($\mathscr{F}$,K).

*Proof.* Due to Prop. 4.2, $\exists E \in \mathscr{E}, E \models$ K-LINEARISABLE($\mathscr{F}$,K) $\wedge\, E \not\models$ STRONGEVENTUAL($\mathscr{F}$). Thus, because of Def. 3.3, STRONGEVENTUAL($\mathscr{F}$) $\not\rightarrow$ K-LINEARISABLE($\mathscr{F}$,K). □

- PREFIXSEQUENTIAL($\mathscr{F}$) and stronger models.

  PREFIXSEQUENTIAL($\mathscr{F}$) ensures eventual convergence because it uses a primary manager per set of objects that imposes a common commit write order among all processes. Therefore, STRONGEVENTUAL($\mathscr{F}$) is weaker than PREFIXSEQUENTIAL($\mathscr{F}$), as stated in the following theorem.

  **Theorem 4.8.** STRONGEVENTUAL*($\mathscr{F}$)* $\rightarrow$ PREFIXSEQUENTIAL*($\mathscr{F}$).*

  *Proof.* In LAZYSINGLEORDER, each time a write $op$ becomes committed in an execution $E$, $op$ is conceptually complete and it becomes visible to other processes. Then, it is included in both $E.vis$ and $E.ar$ relations, in a way that ensures that $E.vis = E.ar$. Thus, LAZYSINGLEORDER $\Rightarrow$ PEROBJECTSINGLEORDER. Therefore, Lemmas 4.3, 4.4 and 4.5 imply that: LAZYSINGLEORDER $\wedge$ RVAL($\mathscr{F}$) $\Rightarrow$ STRONGCONVERGENCE $\wedge$ EVENTUALVISIBILITY $\wedge$ NOCIRCULARCAUSALITY $\wedge$ RVAL($\mathscr{F}$). MONOTONICWRITES is not a relaxing predicate in this scope, since it constrains *vis*. Therefore, according to Def. 3.3, STRONGEVENTUAL($\mathscr{F}$) $\rightarrow$ PREFIXSEQUENTIAL($\mathscr{F}$). □

  **Corollary 4.7.** STRONGEVENTUAL*($\mathscr{F}$)* $\rightarrow$ REGULAR*($\mathscr{F}$).*

  *Proof.* REGULAR($\mathscr{F}$) is stronger than PREFIXSEQUENTIAL($\mathscr{F}$). Relation "*weaker than*" (Def. 3.3) is transitive. So, due to Theorem 4.8, STRONGEVENTUAL($\mathscr{F}$) is weaker than REGULAR($\mathscr{F}$). □

- Other models.

  A single CAP-constrained model remains unexplored in the CAP-constrained vs CAP-free frontier: TIMEDCAUSAL($\mathscr{F}, \Delta$). Let us assess it in this theorem.

  **Theorem 4.9.** STRONGEVENTUAL*($\mathscr{F}$)* $\not\rightarrow$ TIMEDCAUSAL*($\mathscr{F}, \Delta$).*

  *Proof.* Let assume the following system and execution: $S = (\{p_1, p_2\}, \{x\})$, $\Delta = 2$, $\exists E_6 \in \mathscr{E}^{TimedCausal} : E_6 = (\{o_1 = (p_1, wr, x, 1, \sqcup, 0, 1), o_2 = (p_2, wr, x, 2, \sqcup, 0, 1), o_3 = (p_1, rd, x, \sqcup, 1, 1, 2), o_4 = (p_2, rd, x, \sqcup, 2, 1, 2), o_5 = (p_1, rd, x, \sqcup, 2, 3, 4), o_6 = (p_2, rd, x, \sqcup, 1, 3, 4)\}, \{(o_1, o_3), (o_3, o_5), (o_2, o_4), (o_4, o_6), (o_2, o_5), (o_1, o_6)\}, \{(o_4, o_1), (o_1, o_6), (o_6, o_3), (o_3, o_2), (o_2, o_5)\})$.

  $E_6 \models$ TIMEDCAUSAL($\mathscr{F}$,2), and $vis^{-1}(o_5) \mid_{wr} = vis^{-1}(o_6) \mid_{wr} = \{o_1, o_2\}$. However, $o_5.oval \neq o_6.oval$. Therefore, $E_6 \not\models$ STRONGEVENTUAL($\mathscr{F}$). This means that, $\mathscr{E}^{TimedCausal} \not\subset \mathscr{E}^{StrongEventual}$. Thus, according to Def. 3.3, STRONGEVENTUAL($\mathscr{F}$) $\not\rightarrow$ TIMEDCAUSAL($\mathscr{F}, \Delta$). □

The focus of this analysis has been set on some models whose strength suggested that might be convergent. Some notes may be also given about other CAP-free models. In that area, Lloyd et al. [22] prove that CAUSAL+($\mathscr{F}$) is stronger than STRONGEVENTUAL($\mathscr{F}$) and Mahajan et al. [9] prove that REALTIMECAUSAL($\mathscr{F}$) also implies STRONGEVENTUAL($\mathscr{F}$). It is also known that CAUSAL($\mathscr{F}$) is not convergent [22]. So, $\exists E_7 \models$ CAUSAL($\mathscr{F}$) : $E_7 \not\models$ STRONGEVENTUAL($\mathscr{F}$), then STRONGEVENTUAL($\mathscr{F}$) $\not\rightarrow$ CAUSAL($\mathscr{F}$). By Def. 3.3, every model $M_w$ weaker than CAUSAL($\mathscr{F}$) will not be stronger than STRONGEVENTUAL($\mathscr{F}$), since $E_7 \models M_w$ and $E_7 \not\models$ STRONGEVENTUAL($\mathscr{F}$). Therefore, the *causal*, *PRAM*, *RYW*, *MW*, *MR* and *slow* models are not stronger than *strong eventual*. This means that they are not inherently convergent. Fork-based models do not comply either with the

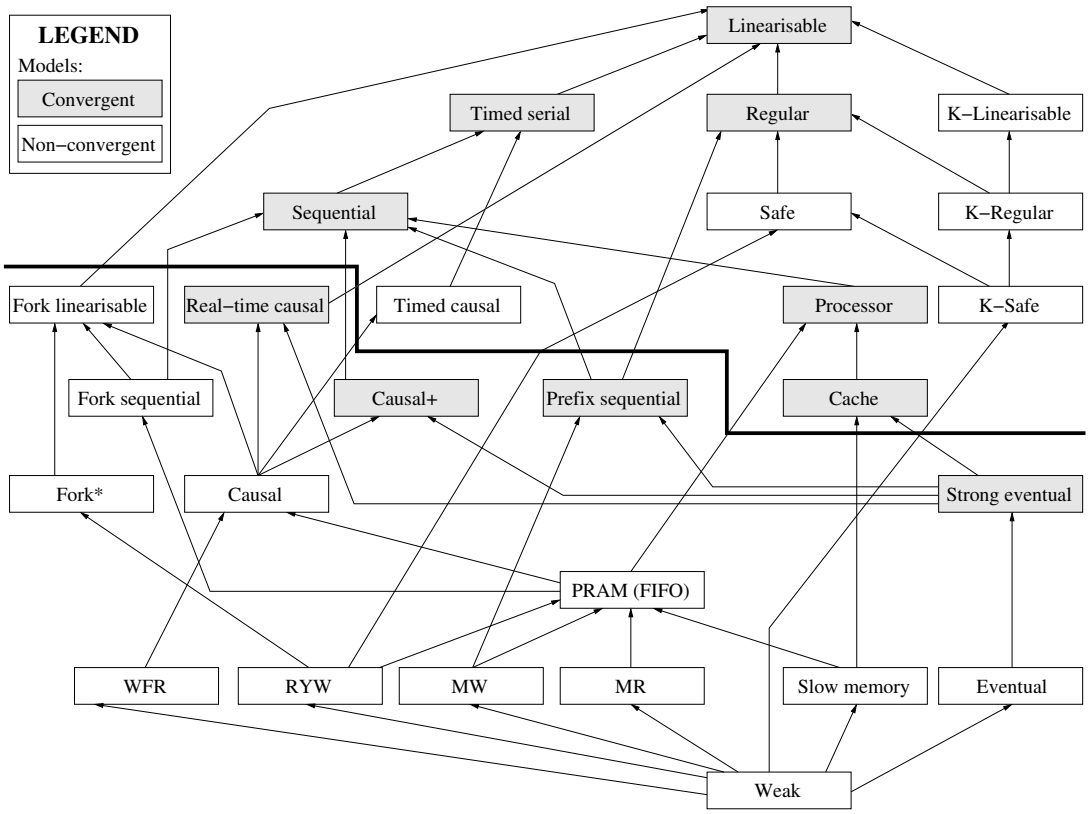


Figure 4: Revised "*weaker than*" ($\rightarrow$) relationships among models.

STRONGEVENTUAL($\mathscr{F}$) definition since their NOJOIN (or ATMOSTONEJOIN) predicate explicitly prevents EVENTUALVISIBILITY from becoming true.

Figure 4 shows the *weaker than* relations among STRONGEVENTUAL($\mathscr{F}$) and other models. The bold line depicts the frontier presented in Figure 2. Models placed above that frontier are CAP-constrained, while the others are CAP-free. Inherently convergent models are shown in grey boxes, while the others are in white boxes.

As shown in Figure 4, our analysis distributes consistency models in these classes regarding convergence:

1. CAP-constrained and not necessarily convergent models: TIMEDCAUSAL($\mathscr{F},\Delta$), K-LINEARISABLE($\mathscr{F},K$), K-REGULAR($\mathscr{F},K$), SAFE($\mathscr{F}$), K-SAFE($\mathscr{F},K$), ... These models lose availability when a network partition arises and do not ensure convergence when no network partition has occurred. This set is the worst one regarding state convergence and CAP freedom.

2. CAP-constrained and convergent models: LINEARISABLE($\mathscr{F}$), TIMEDSERIAL($\mathscr{F},\Delta$), SEQUENTIAL($\mathscr{F}$), PROCESSOR($\mathscr{F}$), CACHE($\mathscr{F}$),... These models lose availability when a network partition arises, but they keep state convergence among processes when no network partition occurs.

   In spite of this, the availability loss may be overcome in some of these models. To this end, the *locality* property may be used. Herlihy and Wing [15] stated that a consistency model $M$ is local when an implementation of $M$ for disjoint subsets of objects is an implementation of $M$ for the whole set of objects. LINEARISABLE($\mathscr{F}$) and CACHE($\mathscr{F}$) are local models [15, 41]. This means that when a service has been implemented respecting any local model $M_l$ and it is deployed in a way that each node group holds a disjoint subset of the service objects (using to this end a database sharding approach [42]), if the network becomes partitioned, every node group still remains available to its users and the whole system still complies with $M_l$. In practice, this means that the system tolerates those network partitions while it remains available, despite

using one of those CAP-constrained and convergent models. The key for ensuring this is to have a good data sharding approach, and there has been a fruitful line of research in that area [43, 44, 45, 46, 47, 48, 49].

3. CAP-free and convergent models: STRONGEVENTUAL($\mathscr{F}$), CAUSAL+($\mathscr{F}$), REALTIMECAUSAL($\mathscr{F}$), PREFIXSEQUENTIAL($\mathscr{F}$),... These models keep availability when a network partition arises and they are able to reach convergence among all processes when there is no connectivity problem or connectivity is recovered. This is one of the most interesting sets, since these models are able to overcome all CAP constraints without indefinitely losing state convergence.

4. CAP-free and not necessarily convergent models: WEAK($\mathscr{F}$), PRAM($\mathscr{F}$), EVENTUAL($\mathscr{F}$), CAUSAL($\mathscr{F}$), ... These models keep availability in all processes when a network partition occurs, and this is a good characteristic, but to this end they do not ensure state convergence. They are generally considered too relaxed to implement and deploy scalable services.

# 5 Related Work

The identification of the set of consistency models affected by the CAP theorem has been implicitly undertaken by several recent papers that have looked for the strongest consistency to be supported in available and partition-tolerant systems [22, 9, 11]. Those papers have taken as a base causal consistency, adding some conditions in order to strengthen it, generating in that way the *causal+* [22], *real-time causal* [9] and *observable causal* [11] models. Those models are incomparable to each other and they define part of the strongest subset of models in the CAP-free set.

Another traditional approach to implement available and partition-tolerant services is based on *eventual* consistency [50]. The term *eventual consistency* was probably first used in the Clearinghouse system [16] and in the Lotus/Iris Notes project on computer-supported cooperative work [51], in 1987 and 1988, respectively. However, such kind of consistency was already explained and used in other previous papers, being the works from Johnson and Thomas [4, 52] (1975) and the commutative-update replication protocol variant of Alsberg and Day [53] (1976) the first ones we are aware of.

Thus, both causal and eventual consistencies belong to the CAP-free set of models. Causal consistency does not demand consensus on a common order of writes, while eventual consistency relaxes the recency of the values being read since it uses lazy write propagation. Intuitively, this suggests that CAP-constrained models are those requiring either consensus on a global write-order (impossible to attain in a partitionable system due to the FLP impossibility result [54]) or a close to immediate recency on the read values (broken when the latest values have been written in another network component while the network is partitioned). However, that conjecture had not been explicitly proven yet.

The exact frontier is not easy to set. Previous attemps were centred on value-order consensus, e.g. [10], and they only provided a partial frontier. The consistency specification framework proposed by Burckhardt et al. [13, 14] provides an excellent basis for specifying consistency models. With it, it is easy to state both safety and liveness conditions. Viotti and Vukolić [12] have used that framework for surveying distributed consistency models. Our work complements their survey in regard to the identification of multiple weaker-than relations between models that were not depicted in [12] and in looking for the CAP-constrained to CAP-free frontier considering value-order consensus and read recency criteria.

To this end, considering the formal models from [13, 14, 12], we have been able to state when a model is CAP-constrained or CAP-free. Besides, we have also proven two interesting propositions: (1) if a model A is weaker than another CAP-free model B, then A is CAP-free, and (2) if a model C is stronger than another CAP-constrained model D, then C is CAP-constrained. Those two properties, combined with our extended hierarchy of models based on the weaker-than relation, facilitates the determination of any newly proposed consistency model as either CAP-free or CAP-constrained.

There are several composable and tunable consistency models [34, 35, 36, 37, 38, 33, 39, 40] whose implementation protocols support both CAP-constrained and CAP-free consistencies. They are a good choice to overcome the limitations imposed by the CAP theorem on the consistency of highly available distributed services. Services that use those models may choose a configuration that either relaxes consistency or easily determines which service activities may remain blocked while partitions arise.

Additionally, other configurations may provide quite a strong consistency while the network remains connected.

However, there may be some problem domains (e.g., cryptocurrency supporting protocols) where their potentially huge request arrival rates and their large set of participating servers may prevent those servers from directly communicating to each other each time a request is processed. In those cases, a CAP-free and eventually convergent model may be the best choice (e.g., *prefix sequential*) at every time, even when no communication failure arises [32]. In those scenarios, the goal is to find an inherently convergent model that reduces to a minimum the interaction among servers, using to this end decentralised algorithms. This shows that not all problems require hybrid tunable models that choose a CAP-constrained variant in connected intervals and a CAP-free one when a disconnection occurs. Indeed, *prefix sequential* guarantees *sequential* consistency when all committed writes are known by every process.

# 6 Conclusions

The CAP theorem was originally proved considering *linearisable* consistency, but there are other consistency models that cannot guarantee replicated service availability when the interconnecting network remains partitioned. All these models are CAP-constrained. In order to find out which is the whole set of CAP-constrained models a precise specification of that concept is needed. That specification has been provided in this paper, accompanied by that of CAP-free models.

With those definitions, a precise frontier between the CAP-free and CAP-constrained sets of consistency models has been determined. Besides, it has been proved that: (1) all models that are weaker than a CAP-free model are also CAP-free, and (2) all models that are stronger than a CAP-constrained model are also CAP-constrained. Those two propositions and the CAP-free vs CAP-constrained frontier provide a basis for easily determining whether any new consistency model A is either CAP-free or CAP-constrained. To this end, A should be compared with any strong CAP-free model F (being CAP-free if A is weaker than F) or with any weak CAP-constrained model C (being CAP-constrained if A is stronger than C).

In order to facilitate the assessment of any new model, we have revised the weaker-than relations among consistency models, refining the hierarchy identified by Viotti and Vukolić in [12]. In this scope, our analysis has been centred in the StrongConvergence property. Thus, we have found that not all CAP-constrained models are strongly convergent, while there are a few CAP-free models that are inherently convergent. Convergent CAP-free models (e.g., *causal+*, *real-time causal*, *observable causal*, *prefix sequential*...) allow service availability while the network is partitioned and easily reach convergence once network connectivity is resumed, overcoming in this way all constraints imposed by the CAP theorem. Considering our resulting hierarchy of models, and the propositions mentioned before, every forthcoming (and potentially convergent and CAP-free) consistency model will be easily characterised once its location in this hierarchy is found.

# References


[1] Davidson, S. B., García-Molina, H., and Skeen, D. (1985) Consistency in partitioned networks. *ACM Comput. Surv.*, **17**, 341–370.

[2] Fox, A. and Brewer, E. A. (1999) Harvest, yield and scalable tolerant systems. *7th Workshop on Hot Topics in Operating Systems (HotOS)*, Rio Rico, Arizona, USA, 28-30 March, pp. 174–178. IEEE-CS Press, Los Alamitos, CA, USA.

[3] Gilbert, S. and Lynch, N. (2002) Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, **33**, 51–59.

[4] Johnson, P. R. and Thomas, R. H. (1975). The maintenance of duplicate databases. RFC 677, Network Working Group, Internet Engineering Task Force.

[5] Birman, K. P. and Friedman, R. (1996) Trading consistency for availability in distributed systems. Technical report. 96-1579, Department of Computer Science, Cornell University, Ithaca, NY, USA.

[6] Lamport, L. (1986) On interprocess communication. Part II: algorithms. *Distrib. Comput.*, **1**, 86–101.

[7] Muñoz-Escoí, F. D. and Bernabéu-Aubán, J. M. (2017) A survey on elasticity management in PaaS systems. *Computing*, **99**, 617–656.

[8] Brewer, E. A. (2012) CAP twelve years later: How the "rules" have changed. *IEEE Comput.*, **45**, 23–29.

[9] Mahajan, P., Alvisi, L., and Dahlin, M. (2011) Consistency, availability and convergence. Technical report. UTCS TR-11-22, Department of Computer Science, The University of Texas at Austin, USA.

[10] Pascual-Miret, L., González de Mendívil, J. R., Bernabéu-Aubán, J. M., and Muñoz-Escoí, F. D. (2015) Widening CAP consistency. Technical report. IUMTI-SIDI-2015/03, Inst. Univ. Mixto Tecnológico de Informática, Univ. Politècnica de València, Valencia, Spain.

[11] Attiya, H., Ellen, F., and Morrison, A. (2017) Limitations of highly-available eventually-consistent data stores. *IEEE Trans. Parallel Distrib. Syst.*, **28**, 141–155.

[12] Viotti, P. and Vukolić, M. (2016) Consistency in non-transactional distributed storage systems. *ACM Comput. Surv.*, **49**, 19:1–19:34.

[13] Burckhardt, S., Gotsman, A., Yang, H., and Zawirski, M. (2014) Replicated data types: specification, verification, optimality. *41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, San Diego, CA, USA, 20-21 January, pp. 271–284. ACM Press, New York, NY, USA.

[14] Burckhardt, S. (2014) Principles of eventual consistency. *Foundations Trends Program. Lang.*, **1**, 1–150.

[15] Herlihy, M. and Wing, J. M. (1990) Linearizability: A correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.*, **12**, 463–492.

[16] Demers, A. J., Greene, D. H., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H. E., Swinehart, D. C., and Terry, D. B. (1987) Epidemic algorithms for replicated database maintenance. *6th ACM Symposium on Principles of Distributed Computing (PODC)*, Vancouver, British Columbia, Canada, 10-12 August, pp. 1–12. ACM Press, New York, NY, USA.

[17] Shapiro, M., Preguiça, N. M., Baquero, C., and Zawirski, M. (2011) Conflict-free replicated data types. *13th Int. Symp. Stabilization, Safety, and Security of Distributed Systems (SSS)*, Grenoble, France, 10-12 October, pp. 386–400. Springer, Berlin, Germany.

[18] Lipton, R. J. and Sandberg, J. S. (1988) PRAM: A scalable shared memory. Technical report. CS-TR-180-88, Princeton University, Princeton, NJ, USA.

[19] Lamport, L. (1979) How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE T. Comput.*, **28**, 690–691.

[20] Terry, D. B., Demers, A. J., Petersen, K., Spreitzer, M., Theimer, M., and Welch, B. B. (1994) Session guarantees for weakly consistent replicated data. *3rd Int. Conf. Parallel and Distributed Information Systems (PDIS)*, Austin, Texas, USA, 28-30 September, pp. 140–149. IEEE-CS Press, Los Alamitos, CA, USA.

[21] Ahamad, M., Burns, J. E., Hutto, P. W., and Neiger, G. (1991) Causal memory. *5th Int. Workshop on Distributed Algorithms and Graphs (WDAG)*, Delphi, Greece, 7-9 October, pp. 9–30. Springer, Berlin, Germany.

[22] Lloyd, W., Freedman, M. J., Kaminsky, M., and Andersen, D. G. (2011) Don't settle for eventual: scalable causal consistency for wide-area storage with COPS. *23rd ACM Symp. Operating Systems Principles (SOSP)*, Cascais, Portugal, 23-26 October, pp. 401–416. ACM Press, New York, NY, USA.

[23] Terry, D. B., Theimer, M., Petersen, K., Demers, A. J., Spreitzer, M., and Hauser, C. (1995) Managing update conflicts in Bayou, a weakly connected replicated storage system. *15th ACM Symposium on Operating System Principles (SOSP)*, Copper Mountain Resort, Colorado, USA, 3-6 December, pp. 172–183. ACM Press, New York, NY, USA.

[24] Torres-Rojas, F. J. and Meneses, E. (2005) Convergence through a weak consistency model: Timed causal consistency. *CLEI Electron. J.*, **8**, 2:1–2:10.

[25] Torres-Rojas, F. J., Ahamad, M., and Raynal, M. (1999) Timed consistency for shared distributed objects. *18th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, Atlanta, Georgia, USA, 3-6 May, pp. 163–172. ACM Press, New York, NY, USA.

[26] Aiyer, A. S., Alvisi, L., and Bazzi, R. A. (2005) On the availability of non-strict quorum systems. *19th Int. Conf. Distributed Computing (DISC)*, Cracow, Poland, 26-29 September, pp. 48–62. Springer, Berlin, Germany.

[27] Mazières, D. and Shasha, D. E. (2002) Building secure file systems out of byzantine storage. *21st Annual ACM Symp. Principles of Distributed Computing (PODC)*, Monterrey, CA, USA, 21-24 July, pp. 108–117. ACM Press, New York, NY, USA.

[28] Oprea, A. and Reiter, M. K. (2006) On consistency of encrypted files. *20th Int. Symp. Distributed Computing (DISC)*, Stockholm, Sweden, 18-20 September, pp. 254–268. Springer, Berlin, Germany.

[29] Li, J. and Mazières, D. (2007) Beyond one-third faulty replicas in Byzantine fault tolerant systems. *4th Symp. Networked Systems Design and Implementation (NSDI)*, Cambridge, Massachusetts, USA, 11-13 April, pp. 131–144. USENIX, Berkeley, CA, USA.

[30] Hutto, P. W. and Ahamad, M. (1990) Slow memory: Weakening consistency to enhance concurrency in distributed shared memories. *10th Int. Conf. Distributed Computing Systems (ICDCS)*, Paris, France, 28 May-1 June, pp. 302–309. IEEE-CS Press, Los Alamitos, CA, USA.

[31] Goodman, J. R. (1989) Cache consistency and sequential consistency. Technical report. Number 61, IEEE Scalable Coherent Interface Working Group, Stanford, CA, USA.

[32] Girault, A., Gößler, G., Guerraoui, R., Hamza, J., and Seredinschi, D. (2018) Monotonic prefix consistency in distributed systems. *38th Int. Conf. Formal Techniques for Distributed Objects, Components, and Systems (FORTE)*, Madrid, Spain, 18-21 June, pp. 41–57. Springer, Berlin, Germany.

[33] Bailis, P., Venkataraman, S., Franklin, M. J., Hellerstein, J. M., and Stoica, I. (2012) Probabilistically bounded staleness for practical partial quorums. *PVLDB*, **5**, 776–787.

[34] Attiya, H. and Friedman, R. (1992) A correctness condition for high-performance multiprocessors. *24th Annual ACM Symposium on Theory of Computing (STOC)*, Victoria, British Columbia, Canada, 4-6 May, pp. 679–690. ACM Press, New York, NY, USA.

[35] Ladin, R., Liskov, B., Shrira, L., and Ghemawat, S. (1992) Providing high availability using lazy replication. *ACM Trans. Comput. Syst.*, **10**, 360–391.

[36] Yu, H. and Vahdat, A. (2002) Design and evaluation of a conit-based continuous consistency model for replicated services. *ACM Trans. Comput. Syst.*, **20**, 239–282.

[37] Krishnamurthy, S., Sanders, W. H., and Cukier, M. (2002) An adaptive framework for tunable consistency and timeliness using replication. *Int. Conf. Dependable Systems and Networks (DSN)*, Bethesda, MD, USA, 23-26 June, pp. 17–26. IEEE-CS Press, Los Alamitos, CA, USA.

[38] Santos, N., Veiga, L., and Ferreira, P. (2007) Vector-field consistency for ad-hoc gaming. *ACM/IFIP/USENIX 8th Int. Middleware Conf.*, Newport Beach, CA, USA, 26-30 November, pp. 80–100. Springer, Berlin, Germany.

[39] Li, C., Porto, D., Clement, A., Gehrke, J., Preguiça, N. M., and Rodrigues, R. (2012) Making geo-replicated systems fast as possible, consistent when necessary. *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Hollywood, CA, USA, 8-10 October, pp. 265–278. USENIX, Berkeley, CA, USA.

[40] Dobre, D., Viotti, P., and Vukolić, M. (2014) Hybris: Robust hybrid cloud storage. *ACM Symposium on Cloud Computing (SoCC)*, Seattle, WA, USA, 3-5 November, pp. 12:1–12:14. ACM Press, New York, NY, USA.

[41] Vitenberg, R. and Friedman, R. (2003) On the locality of consistency conditions. *17th Int. Conf. Distributed Computing (DISC)*, Sorrento, Italy, 1-3 October, pp. 92–105. Springer, Berlin, Germany.

[42] Ceri, S., Negri, M., and Pelagatti, G. (1982) Horizontal data partitioning in database design. *Int. Conf. Management of Data (SIGMOD)*, Orlando, Florida, 2-4 June, pp. 128–136. ACM Press, New York, NY, USA.

[43] Curino, C., Zhang, Y., Jones, E. P. C., and Madden, S. (2010) Schism: a workload-driven approach to database replication and partitioning. *Proc. VLDB Endowment*, **3**, 48–57.

[44] Bernstein, P. A., Cseri, I., Dani, N., Ellis, N., Kalhan, A., Kakivaya, G., Lomet, D. B., Manne, R., Novik, L., and Talius, T. (2011) Adapting Microsoft SQL Server for cloud computing. *27th Int. Conf. Data Engineering (ICDE)*, Hannover, Germany, 11-16 April, pp. 1255–1263. IEEE-CS Press, Los Alamitos, CA, USA.

[45] Liroz-Gistau, M., Akbarinia, R., Pacitti, E., Porto, F., and Valduriez, P. (2012) Dynamic workload-based partitioning for large-scale databases. *23rd Int. Conf. Database and Expert Systems Applications (DEXA)*, Vienna, Austria, 3-6 September, pp. 183–190. Springer, Berlin, Germany.

[46] Das, S., Agrawal, D., and El Abbadi, A. (2013) ElasTraS: An elastic, scalable, and self-managing transactional database for the cloud. *ACM Trans. Database Syst.*, **38**, 5:1–5:45.

[47] Chen, Z., Yang, S., Tan, S., He, L., Yin, H., and Zhang, G. (2015) A new fragment re-allocation strategy for NoSQL database systems. *Frontiers Comput. Sci.*, **9**, 111–127.

[48] Kamal, J., Murshed, M. M., and Buyya, R. (2016) Workload-aware incremental repartitioning of shared-nothing distributed databases for scalable OLTP applications. *Future Generation Comp. Syst.*, **56**, 421–435.

[49] El-Ghamrawy, S. M. and Hassanien, A. E. (2017) A partitioning framework for Cassandra NoSQL database using Rendezvous hashing. *J. Supercomput.*, **73**, 4444–4465.

[50] Muñoz-Escoí, F. D., García-Escrivá, J. R., Sendra-Roig, J. S., Bernabéu-Aubán, J. M., and González de Mendívil, J. R. (2018) Eventual consistency: Origin and support. *Comput. Inform.*, **37**, 1037–1072.

[51] Kawell Jr., L., Beckhardt, S., Halvorsen, T., Ozzie, R., and Greif, I. (1988) Replicated document management in a group communication system. *ACM Conf. Computer-Supported Cooperative Work (CSCW)*, Portland, Oregon, USA, 26-28 September, pp. 395–. ACM Press, New York, NY, USA.

[52] Cosell, B. S., Johnson, P. R., Malman, J. H., Schantz, R. E., Sussman, J., Thomas, R. H., and Walden, D. C. (1975) An operational system for computer resource sharing. *5th ACM Symposium on Operating System Principles (SOSP)*, The University of Texas at Austin, Austin, Texas, USA, 19-21 November, pp. 75–81. ACM Press, New York, NY, USA.

[53] Alsberg, P. and Day, J. D. (1976) A principle for resilient sharing of distributed resources. *2nd Int. Conf. Software Engineering (ICSE)*, San Francisco, CA, USA, 13-15 October, pp. 562–570. IEEE-CS Press, Los Alamitos, CA, USA.

[54] Fischer, M. J., Lynch, N. A., and Paterson, M. (1985) Impossibility of distributed consensus with one faulty process. *J. ACM*, **32**, 374–382.