CAP Theorem:
Revision of its Related Consistency Models

Francesc D. Muñoz-Escoí[1], Rubén de Juan-Marín[1],
J. R. González de Mendívil[2], José M. Bernabéu-Aubán[1]

[1]Instituto Universitario Mixto Tecnológico de Informática
Universitat Politècnica de València
46022 Valencia (Spain)

[2]Depto. de Ingeniería Matemática e Informática
Universidad Pública de Navarra
31006 Pamplona (Spain)

{fmunyoz,rjuan,josep}@iti.upv.es, mendivil@unavarra.es

Technical Report TR-IUMTI-SIDI-2017/002

CAP Theorem: Revision of its Related Consistency Models

Francesc D. Muñoz-Escoí[1], Rubén de Juan-Marín[1],
J. R. González de Mendívil[2], José M. Bernabéu-Aubán[1]

[1]Instituto Universitario Mixto Tecnológico de Informática
Universitat Politècnica de València
46022 Valencia (Spain)

[2]Depto. de Ingeniería Matemática e Informática
Universidad Pública de Navarra
31006 Pamplona (Spain)

Technical Report TR-IUMTI-SIDI-2017/002

e-mail: {fmunyoz,rjuan,josep}@iti.upv.es, mendivil@unavarra.es

July 2017

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 proven assuming that "consistency" refers to atomic consistency. However, multiple memory consistency models exist and atomic consistency is in one of the edges of that spectrum: the strongest one. 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 which is the set of consistency models that are 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 more accurate hierarchy of consistency models depending on their strength is built.

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 (Davidson et al, 1985) long time ago [16], thoroughly explained by Fox and Brewer [22] in 1999 and proved by Gilbert and Lynch [23] in 2002. However, the compromise between strongly consistent actions, availability and tolerance to network partitions was already implicit in Johnson and Thomas (1975) [28] and justified by Birman and Friedman [10] 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 answered in a reasonable time. A service is partition-tolerant if once a network partition arises, dividing the service instances in two (or more) disjoint sets unable to communicate with each other, all those disjoint sets may go on. 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 [37]; 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 Gilbert and Lynch to assume that kind of consistency in their proof of the CAP theorem [23].

With the advent of cloud computing, it is easy to develop and deploy highly scalable distributed services [46]. 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 [11] explains that network partitions are rare, even for world-wide geo-replicated services. If services demand partition tolerance and availability, their consistency may be still quite strong most of the time, relaxing it when any temporary network partition arises.

Therefore, it seems interesting to explore 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. 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, where could we set the border between CAP-constrained and *CAP-free* models (i.e., those that may be supported while both availability and network partition tolerance are guaranteed)? Although some partial answers to these questions have been given in previous papers [44, 50, 6], let us provide a new and revised answer to them in the following sections.

2 System Model

We assume a distributed system $S = (ProcessIds, Network, ObjectIds)$. S is partially synchronous and consists of: (1) a set of processes *ProcessIds*, (2) intercommunicated through a set of links called *Network*, and (3) a set of objects *ObjectIds*. Processes in *ProcessIds* may fail. Scalable distributed services may be deployed in *S*. Those distributed services consist of multiple elements, implemented as a set of objects *ObjectIds*. Objects may be replicated in order to improve their availability. In those cases, their instances are deployed in *ProcessIds* using a replication protocol and respecting some replica consistency model.

A function $Connects : \mathscr{P}(ProcessIds) \times \mathscr{P}(Network) \rightarrow \{true, false, undef\}$ is also needed. It is called as $Connects(P,N)$ using a subset of processes P and an internal network N returns *true* when the internal network N can be used by every process in P, *false* if none of the processes of P may send their messages using N and *undef* otherwise.

Network might fail by becoming divided and generating a temporary network partition. In those cases S is partitioned in a set of K network components such that:

- $\bigcup_{i \in K} S_i \subset S$, where $S_i = (ProcessIds_i, Network_i, ObjectIds)$
- $ProcessIds = \bigcup_{i \in K} ProcessIds_i$
- $\forall i, j \in K, i \neq j : ProcessIds_i \bigcap ProcessIds_j = \emptyset$
- $\bigcup_{i \in K} Network_i \subset Network$
- $\forall i, j \in K, i \neq j : Connects(ProcessIds_i, Network_j) = false$
- $\forall i \in K : Connects(ProcessIds_i, Network_i) = true$

This means that when *Network* is partitioned, multiple disjoint network components $S_i = (ProcessIds_i, Network_i, ObjectIds)$ exist. Processes located in different components cannot communicate with each other. However, processes in the same component intercommunicate without problems.

3 Finding a Consistency Borderline

The first question from the introduction was already partially answered by Birman and Friedman [10], since their proof did not rely on *atomic* consistency. Instead of this, they proved the following result: in a partitioned network "*no set of processes that implement a service in a responsive way can execute strongly non commutative actions concurrently*". The term "*in a responsive way*" is a synonym for "*available*". On the other hand, "*strongly non commutative actions*" refers in that paper to a serial execution of a set of operations. This is the main requirement: considering all replicated objects managed by a set of servers, all replicas should agree on a given sequence of operations. Moreover, that sequence, when restricted to a given process, should be consistent with the local execution of that process. Those are the two conditions that define *sequential consistency* [36]. Therefore, considering [10] there are at least two different CAP-constrained models: *atomic* and *sequential*.

Birman and Friedman [10] gave another hint related to CAP: one way to relax consistency in order to maintain availability in partitioned systems is to rely on commutative operations. Commutative operations allow temporary divergences among isolated components. Once the network partition disappears and connectivity is resumed, those operations must be applied in all other components to reach convergence. But this was not a new result; that fact was already explained in the first papers proposing replication protocols for systems that could become partitioned, as [18], or that tried to improve their performance combining commutativity with a multi-master approach and relaxing the total order of their updates. The latter was already suggested by Alsberg and Day (1976) as a protocol variant in [3].

Commutative (i.e., conflict-free) operations provide the key to answer our second question. In order to break the CAP constraints we need some kinds of operations or consistency that admit continuity even when not all processes in that system can be reached. If there were any consistency models allowing such behaviour, they would be CAP-free. A hint to answer this question was provided by Attiya and Friedman in [5]. They identified "*fast*" consistency models: those that may complete both read and write operations faster than the network delay; i.e., without contacting other processes. This means that the updates caused by a write operation will be known by the other replicas once the write has already returned control to its invoker.

At a glance, writer processes in fast consistency models do not require any kind of synchronisation with the remaining processes. Once the write is locally completed, such operation is considered as terminated. Its delivery time in the remaining processes is unimportant since the local value order at its writer may be different to the value order in the other processes. This means that there is no single global value order in those models. On the other hand, non-fast models usually require communication with other processes for achieving consensus on a unique global value order.

When network partitions arise or processes may fail, the FLP [21] impossibility result states that no consensus is possible among the participating processes. Therefore, every consistency model requiring consensus will not be attainable in a partitionable distributed system. With a similar base, Pascual-Miret et al [50] have proved that the constraints of the CAP theorem apply to all data consistency models requiring a global agreement on the order of writes to each variable. Due to this, the *atomic*, *sequential*, *processor* [24] and *cache* [24] models are CAP-constrained, while *FIFO* [41] and *causal* [1] consistencies, among others, are CAP-free. Although that result points out at least four CAP-constrained models, extending the original CAP theorem, it is not yet an accurate and complete answer for the second question outlined in the introduction of this paper. In [50], only server-centric [55] models have been considered, but there are other types of consistency models that could be analysed.

Let us take the survey of Viotti and Vukolić [61] on non-transactional distributed data consistency models as a basis to refine the current answer draft. To this end, the rest of this section is structured as follows. Section 3.1 outlines the consistency specification framework used in [61]. Section 3.2 defines what is a *CAP-constrained* model. Section 3.3 presents all consistency models identified in [61] and summarises their properties. Later, Section 3.4 analyses which other consistency models are CAP-constrained. With the latter, a more accurate frontier between the CAP-constrained and CAP-free sets of models is defined.

3.1 Specification Framework

Viotti and Vukolić propose a framework for specifying a large set of distributed (non-transactional) data consistency models in [61]. Such framework is based on that presented by Burckhardt et al in [13]. Since the CAP theorem involves software services deployed in distributed systems, it makes sense to consider those consistency models in this scope. Let us summarise the main elements of that framework. Additional details may be found in [61].

As it has been mentioned in Section 2, services consist of processes and objects, whose identifiers belong to the *ProcessIds* and *ObjectIds* sets, respectively. The values that can be maintained in objects belong to the *Values* set. Those processes interact with the existing objects invoking their operations, whose types belong to the *OpTypes* set. The real-time domain is represented by the *Time* set.

Operation calls are represented using *(proc,type,obj,ival,oval,stime,rtime)* tuples, where:

- *proc* $\in$ *ProcessIds* is the identifier of the process that invokes the operation.
- *type* $\in$ *OpTypes* is the operation type; e.g., *wr* for writes and *rd* for reads.
- *obj* $\in$ *ObjectIds* is the identifier of the object being invoked.
- *ival* $\in$ *Values* $\cup\{\sqcup\}$ is the operation input value, or $\sqcup$ in case of a read operation.
- *oval* $\in$ *Values* $\cup\{\sqcup,\nabla\}$ is the operation output value, or $\sqcup$ in case of a write or ∇ if the operation does not return.
- *stime* $\in$ *Time* is the operation invocation time.
- *rtime* $\in$ *Time* is the operation return time.

A history H is a set of operations. A history contains all operations invoked in an execution. $H\ |_{wr}$ (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 on elements of a history are needed:

- *rb* (*returns-before*) is a partial order on H based on real-time precedence: $rb \equiv \{(a,b) : a,b \in H \wedge a.rtime < b.stime\}$.
- *ss* (*same-session*) is an equivalence relation on H that groups the operations invoked by the same process: $ss \equiv \{(a,b) : a,b \in H \wedge a.proc = b.proc\}$.
- *so* (*session order*) is a partial order defined as: $so \equiv rb \cap ss$.
- *ob* (*same-object*) is an equivalence relation on H that groups the operations invoked on the same object: $ob \equiv \{(a,b) : a,b \in H \wedge a.obj = b.obj\}$.
- *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 the function $Concur : H \rightarrow 2^H$ that denotes the set of write operations concurrent with a given operation: $Concur(a) \equiv \{b \in H\ |_{wr}: (a,b) \in concur\}$.

Binary relation projections are denoted as specified in the following example: $rel\ |_{wr\rightarrow rd}$ 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*, and rel^{-1} denotes the inverse relation of *rel*.

An abstract execution is defined as $A = (H, vis, ar)$ and is built on a history H, complemented with two relations *vis* and *ar* on elements of H, where:

- *vis* (*visibility*) is an acyclic relation that accounts for the propagation of write operations; i.e., when $(a,b) \in vis$ then the effects of a are visible to the process that invokes b.

  Two write operations are *invisible* to each other when they are not ordered by *vis*.
- *ar* (*arbitration*) is a total order on operations of the history that specifies how conflicts due to concurrent and invisible operations are resolved in that abstract execution A in order to comply with its intended consistency models. It is a total order because A defines an abstract single sequence of operations.

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

The context C of an operation op in an abstract execution A is defined as: $C = cxt(A, op) \equiv A\mid_{op,vis^{-1}(op),vis,ar}$.

For each replicated data type, a function $\mathscr{F}$ specifies the set of intended return values of an operation $op \in H$ in relation to its context: $\mathscr{F}(op, cxt(A, op))$. With $\mathscr{F}$, the *return value consistency* is defined as: $\text{RVAL}(\mathscr{F}) \equiv \forall op \in H : op.oval \in \mathscr{F}(op, cxt(A, op))$. This predicate guarantees that the return value of any given operation in an execution belongs to the set of its intended return values.

In order to provide an example of $\mathscr{F}$ function, $a = prec(b)$ may be defined as the unique latest operation preceding b in ar, such that $a \in H\mid_{wr} \cap vis^{-1}(b)$. That is, $prec(b)$ is the last write visible to b according to the ar order. With this, a read/write register may be taken as our reference replicated data type, whose $\mathscr{F}$ function is: $\mathscr{F}_{reg}(op, cxt(A, op)) = prec(op).ival$. This means that the output value of a read operation on a register should return the input value of its preceding write operation.

Finally, let us use $A \models \mathscr{P}$ in order to state that the consistency predicate $\mathscr{P}$ is true for abstract execution A. With this, given history H and $\mathscr{A}$ as its set of all possible abstract executions on H, we may say that history H satisfies some consistency predicates $\mathscr{P}_1, \ldots \mathscr{P}_n$ if it can be extended to some abstract execution that satisfies them all: $H \models \mathscr{P}_1 \wedge \ldots \wedge \mathscr{P}_n \Leftrightarrow \exists A \in \mathscr{A} : \mathscr{H}(A) = H \wedge A \models \mathscr{P}_1 \wedge \ldots \wedge \mathscr{P}_n$, where $\mathscr{H}(A)$ denotes H, assuming that $A = (H, vis, ar)$.

3.2 CAP-related Definitions

The goal of this paper consists in delimiting the frontier between two sets of consistency models called *CAP-constrained* and *CAP-free*. Those concepts have been informally introduced in the previous sections. Their definitions are presented hereafter.

Definition 1 (CAP-free consistency model)**.** *A consistency model CM is CAP-free if these two conditions are respected while a network partition happens in the system:*

1. *All correct processes remain available; i.e., all they go on processing object operations.*
2. *The global executions generated by all processes still comply with the consistency predicates that define CM.*

If a consistency model is not CAP-free, then it is CAP-constrained. Therefore:

Definition 2 (CAP-constrained consistency model)**.** *A consistency model CM is CAP-constrained if one of these two conditions is held while a network partition happens in the system:*

1. *If all correct processes remain available, processing object operations, then at least one of the predicates that define CM will not be respected; i.e., CM is no longer respected.*
2. *If CM is respected then its predicates are only maintained by a connected major subset of the live processes. All other processes may still remain alive, but they will be unavailable. This means that those other processes are prevented from executing object operations while the network remains partitioned; i.e., they are kept stopped and from the point of view of CM they have left the system.*

3.3 Distributed Consistency Models

Viotti and Vukolić [61] distinguish ten different groups of consistency models: (1) linearisability 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. The latter (synchronised models) are only described in [61] for completeness since they only make sense in multiprocessor computers but not in a distributed system with message-based communication. Additionally, the models contained in the eighth group cannot be specified with the proposed set of consistency predicates. Because of this, no relationship with the models contained in the other groups can be established for them. We will not consider those two groups hereafter.

In order to summarise the specification of the other eight groups, a set of consistency predicates should be specified before. Table 1 provides those specifications. With them, those groups of models may be defined as described in the following sections.

We informally use the "$\rightarrow$" (*weaker than*) partial order to relate consistency models. Thus, when a model A is weaker than another model B, such fact will be represented as $A \rightarrow B$. This means that the set of executions that comply with B is a proper subset of the set of executions complying with A.

3.3.1 Linearisability and other Strong Models

According to Burckhardt et al [13] *linearisability* [26] may be specified as:

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

Viotti and Vukolić [61] state that *linearisability* is equivalent to *atomic* consistency [37] for read-write registers. Lamport defined in [37] safe and regular registers, besides atomic ones. Those two models are slightly weaker than *atomic* consistency. When no concurrent reads and writes arise in an execution, both safe and regular registers read the last written value, as atomic registers do. However when read and write concurrent actions exist, a safe register may return any value as a result of a read action, while a regular register may return the value of the last complete write or any of the values being written by concurrent writes.

Those models can be specified as follows:

REGULAR$(\mathscr{F}) \equiv$ SINGLEORDER $\wedge$ REALTIMEWRITES $\wedge$ RVAL$(\mathscr{F})$

SAFE$(\mathscr{F}) \equiv$ SINGLEORDER $\wedge$ REALTIMEWRITES $\wedge$ SEQRVAL$(\mathscr{F})$

SEQRVAL$(\mathscr{F})$ is a weaker property than RVAL$(\mathscr{F})$, since it drops read action determinism in case of concurrent writes. REALTIMEWRITES is weaker than REALTIME since it reduces the set of *returns-before* pairs of actions to be considered by the *ar* relation. Thus, the already presented models may be ordered as: SAFE $\rightarrow$ REGULAR $\rightarrow$ LINEARISABILITY.

3.3.2 Weak and Eventual Consistency

Viotti and Vukolić [61] consider that the *weak* consistency corresponds to a model where no consistency property is respected in executions. Therefore, such model is respected even when no synchronisation effort is made by the processors sharing or replicating a set of objects.

Eventual consistency [18, 56, 51, 62] is slightly stronger than the *weak* model and, under its umbrella, we could place many of the first systems that tried to overcome the constraints imposed by network partitions on the availability and consistency of data [28, 3]. This consistency requires that replicas converge to identical copies in the absence of new updates. A formal definition of *eventual* consistency is given by Burckhardt et al [13] as: EVENTUALCONSISTENCY$(\mathscr{F}) \equiv$ EVENTUALVISIBILITY $\wedge$ NOCIRCULAR-CAUSALITY $\wedge$ RVAL$(\mathscr{F})$.

Shapiro et al [54] define a complementary property for *eventual* consistency, called *strong convergence* (already specified in Table 1): all correct replicas that have applied the same write operations have equivalent states. This does not compel processes to apply those write operations in the same order. The name of the resulting consistency is *strong eventual* consistency and can be specified as: STRONGEVENTUAL-CONSISTENCY$(\mathscr{F}) \equiv$ EVENTUALCONSISTENCY$(\mathscr{F}) \wedge$ STRONGCONVERGENCE.

With these specifications, the relationships between the models in this group and those in the previous group may be shown as follows:

WEAK $\rightarrow$ SAFE

WEAK $\rightarrow$ EVENTUALCONSISTENCY $\rightarrow$ STRONGEVENTUALCONSISTENCY $\rightarrow$ LINEARISABILITY

3.3.3 PRAM and Sequential Consistency

The third group of consistency models is based on Pipeline RAM consistency [41], also known as *PRAM* or *FIFO* consistency. *PRAM* requires that all processes see write operations issued by a given process in the same order as they were applied by that process. On the other hand, processes have freedom for interleaving the writes generated by other processes.

PRAM may be defined as: $PRAM \equiv so \subseteq vis$.

Another model stricter than *PRAM* is the *sequential* [36] one. In sequential consistency, all operations are serialised in the same order by all replicas, and the order of writes generated by each process is respected. Thus, its definition is: $\textsc{SequentialConsistency}(\mathscr{F}) \equiv \textsc{SingleOrder} \wedge \text{PRAM} \wedge \textsc{RVal}(\mathscr{F})$.

Since PRAM is weaker than REALTIME, the following relations exist:
$\textsc{Weak} \rightarrow \text{PRAM} \rightarrow \textsc{SequentialConsistency} \rightarrow \textsc{Linearisability}$

3.3.4 Session Guarantees

Session guarantees were proposed by Terry et al [56]. They characterise client sessions using four complementary guarantees:

- *Monotonic reads* (MR) require that successive reads requested by a given process reflect a non-decreasing set of writes. This means that when a process has read a value *v* from an object, any subsequent read operation will not return any value written before *v*. Therefore, its specification demands that the transitive closure of *vis* and *so* is included in *vis*. Formally: $\textsc{MonotonicReads} \equiv \forall a \in H, \forall b,c \in H\mid_{rd}: (a,b) \in vis \wedge (b,c) \in so \Rightarrow (a,c) \in vis$.

- *Read-your-writes* (RYW) requires that a read from a process P can only be served by replicas that have already applied all writes previously invoked by P. Formally: $\textsc{ReadYourWrites} \equiv \forall a \in H\mid_{wr}, \forall b \in H\mid_{rd}: (a,b) \in so \Rightarrow (a,b) \in vis$.

- *Monotonic writes* (MW) demands that a write can be only served in a replica if that replica has already performed all previous writes in the same session. Formally: $\textsc{MonotonicWrites} \equiv \forall a,b \in H\mid_{wr}: (a,b) \in so \Rightarrow (a,b) \in ar$.

- *Writes-follow-reads* (WFR) ensures that writes made in a session are ordered after any writes made by any process on any object whose effects were seen by previous reads in the same session. Formally: $\textsc{WritesFollowReads} \equiv \forall a,c \in H\mid_{wr}, \forall b \in H\mid_{rd}: (a,b) \in vis \wedge (b,c) \in so \Rightarrow (a,c) \in ar$.

This formal specification of session guarantees has been proposed by Viotti and Vukolić [61]. In it, those conditions assume that processes are regular servers. The original specification [56] given by Terry et al was a bit different, since the effects of each condition should apply to the actions of each client process. That difference makes the original session guarantees much stronger than those shown here, but incomparable to other consistency models. Due to this incomparability constraint, let us go on using the specification from Viotti and Vukolić.

Thus, the relationships between session guarantees and the consistency models defined in the previous sections can be summarised as follows [61]:
$\textsc{Weak} \rightarrow \textsc{MonotonicReads} \rightarrow \text{PRAM}$
$\textsc{Weak} \rightarrow \textsc{MonotonicWrites} \rightarrow \text{PRAM}, \textsc{Safe}$
$\textsc{Weak} \rightarrow \textsc{ReadYourWrites} \rightarrow \text{PRAM}, \textsc{Safe}$
$\textsc{Weak} \rightarrow \textsc{WritesFollowReads} \rightarrow \textsc{Sequential}$

3.3.5 Causal Models

Causal consistency models are based on the *happens-before* relation defined by Lamport [35]. Thus, two operations *a* and *b* are potentially causally related ($a \rightarrow b$) when: (i) *a* and *b* have been executed by the same process in that order, or (ii) *b* is a read that returns the value written in *a*, or (iii) they are related by the transitive closure of (i) and (ii). The specification of *causal* consistency was proposed by Ahamad et al [1] and that consistency may be defined as: $\textsc{Causality}(\mathscr{F}) \equiv \textsc{CausalVisibility} \wedge \textsc{CausalArbitration} \wedge \textsc{RVal}(\mathscr{F})$.

Lloyd et al [42] strengthened *causal* consistency requiring that all replicas should reach an agreement in case of write conflicts; i.e., in case of concurrent writes on the same objects. The resulting model is called *convergent causal* consistency, although it is also known as *causal+*. Formally: $\textsc{Causal+}(\mathscr{F}) \equiv \textsc{Causality}(\mathscr{F}) \wedge \textsc{StrongConvergence}$.

Also in 2011, Mahajan et al [44] proposed another way for strengthening causal consistency: the *real-time causal* model. In the latter, according to Viotti and Vukolić, causally concurrent write operations that do not overlap in real time should be applied following their real-time order. Thus, the formal specification of this model is [61]: $\text{REALTIMECAUSALITY}(\mathscr{F}) \equiv \text{CAUSALITY}(\mathscr{F}) \land \text{REALTIME}$.

However, the original specification given in [44] is a bit more relaxed than that. Instead of REALTIME, Mahajan et al propose this condition:

$\text{NOBACKWARDTIME} \equiv \forall a,b \in H : a.rtime < b.stime \Rightarrow (b,a) \notin vis$

They only require that *vis* does not contradict *rb*, but do not force all operation pairs in *rb* be in *vis*. Thus, when $(a,b) \in rb \land (a,b) \notin hb \land (b,a) \notin hb$ then *a* and *b* may still be considered concurrent.

Since NOBACKWARDTIME does not imply STRONGCONVERGENCE (nor the opposite), the *causal+* and *real-time causal* models are incomparable. Thus, the following relations exist within the models contained in this group and among them and the models already presented in previous sections:

$\text{CAUSALITY} \rightarrow \text{CAUSAL+}$
$\text{CAUSALITY} \rightarrow \text{REALTIMECAUSALITY}$
$\text{WRITESFOLLOWREADS} \rightarrow \text{CAUSALITY}$
$\text{PRAM} \rightarrow \text{CAUSALITY}$
$\text{CAUSAL+} \rightarrow \text{SEQUENTIAL}$
$\text{REALTIMECAUSALITY} \rightarrow \text{LINEARISABILITY}$

3.3.6 Staleness-based Models

Staleness-based models allow read operations to return stale written values. They provide stronger guarantees than *eventual* consistency, allowing efficient implementations much faster than *linearisable* consistency. Two general metrics are used for measuring staleness: real time and data object versions.

A first type of time-related staleness was introduced by Terry et al [57] in their proposal of *prefix* consistency. In it, readers observe an ordered sequence of writes that may not contain the effects of the most recent write operations. Thus, it ensures agreement on order but does not provide any guarantee on recency. Taking this model as a base, Viotti and Vukolić [61] propose two complementary consistency models, depending on their level of recency being maintained: *prefix sequential* and *prefix linearisable*. Formally:

$\text{PREFIXSEQUENTIAL}(\mathscr{F}) \equiv \text{SINGLEORDER} \land \text{MONOTONICWRITES} \land \text{RVAL}(\mathscr{F})$
$\text{PREFIXLINEARISABLE}(\mathscr{F}) \equiv \text{SINGLEORDER} \land \text{REALTIMEWW} \land \text{RVAL}(\mathscr{F})$

Viotti and Vukolić [61] define the $\text{TIMEDVISIBILITY}(\Delta)$ predicate (shown in Table 1) for specifying a second type of real-time-based staleness. With it, they provide a short specification for both the *timed causal* [58] and *timed serial* [59, 60] consistencies defined by Torres-Rojas et al. Those models allow a parametrised degree Δ of staleness on the *causal* and *sequential* models. This means that a write operation should be visible to every process in the system in Δ time units. Since both the original *causal* and *sequential* models do not place any bound on write propagation time nor read staleness, both are strengthened with that addition. As a result, *timed causal* is closer to *sequential* consistency than the original *causal* model, while *timed serial* is closer to *linearisability* than the original *sequential* model. They are formally defined, respectively, as follows:

$\text{TIMEDCAUSALITY}(\mathscr{F},\Delta) \equiv \text{CAUSALITY}(\mathscr{F}) \land \text{TIMEDVISIBILITY}(\Delta)$
$\text{TIMEDSERIAL}(\mathscr{F},\Delta) \equiv \text{SINGLEORDER} \land \text{TIMEDVISIBILITY}(\Delta) \land \text{RVAL}(\mathscr{F})$

Aiyer et al [2] propose a set of version-based staleness-tolerant consistency models. Those models are called *K-safe*, *K-regular* and *K-atomic* (or K-linearisability). They take as a base the *safe*, *regular* and *atomic* register-based models defined by Lamport [37], allowing reads that do not overlap concurrent writes to return one of the latest K written values. Thus, these models need the SINGLEORDER predicate for totally ordering the write operations but tolerate stale reads using other more relaxed predicates to complement it. For instance, a formal specification of K-linearisability is:

$\text{K-LINEARISABILITY}(\mathscr{F},K) \equiv \text{SINGLEORDER} \land \text{REALTIMEWW} \land \text{K-REALTIMEREADS}(K) \land \text{RVAL}(\mathscr{F})$

3.3.7 Fork-based Models

Services whose deployment demands the usage of outsourced storage risk to find arbitrary [38] failures that may compromise their execution. Thus, systems that deal with untrusted storage may provide *linearisability* when the storage behaves correctly, but degrade to weaker consistency models when the storage shows a Byzantine [38] fault. *Fork-based* [45] consistency tries to model such behaviour.

Mazières and Shasha [45] proposed the first *fork* (or fork-linearisable) consistency protocol in 2002. In it, the system ensures that when the storage causes the visible histories of two processes to differ (even in only one operation), they may never again read each other's writes without the server being exposed as faulty. Formally:

$\textsc{ForkLinearisability}(\mathscr{F}) \equiv \text{PRAM} \land \textsc{RealTime} \land \textsc{NoJoin} \land \textsc{RVal}(\mathscr{F})$

Several relaxations of that model were proposed thereafter. For instance, Oprea and Reiter [48] required that whenever an operation becomes visible to several processes, all they share the same history of operations occurring before that operation. The resulting model, *fork-sequential*, is defined as follows[61]:

$\textsc{ForkSequential}(\mathscr{F}) \equiv \text{PRAM} \land \textsc{NoJoin} \land \textsc{RVal}(\mathscr{F})$

A second relaxation is the *fork** [40] model proposed by Li and Mazières. In *fork** consistency, forked groups of processes may observe at most one common operation issued by a given correct process. Formally:

$\textsc{Fork}^*(\mathscr{F}) \equiv \textsc{ReadYourWrites} \land \textsc{RealTime} \land \textsc{AtMostOneJoin} \land \textsc{RVal}(\mathscr{F})$

3.3.8 Per-object Models

Per-object orderings allow for more efficient implementations than global orderings, although their resulting consistency is also more relaxed than the latter. There have been multiple consistency models that consider a per-object ordering.

To begin with, *slow memory* was defined by Hutto and Ahamad [27] as a weaker variant of the *PRAM* model. Thus, in the *slow* model all processes see the writes of a given process to a given object in the same order. It may be qualified as a "per-object PRAM" [61]. Formally:

$\textsc{PerObjectPRAM} \,/\, \textsc{Slow} \equiv (so \cap ob) \subseteq vis$

A model slightly stronger is *cache* consistency, defined by Goodman [24]. *Cache* consistency requires agreement on the sequence of writes applied to each object, complemented with the *slow* model semantics. Viotti and Vukolić refer to this model as "per-object sequential", providing this formal definition for it:

$\textsc{PerObjectSequential}(\mathscr{F}) \equiv \textsc{PerObjectSingleOrder} \land \textsc{PerObjectPRAM} \land \textsc{RVal}(\mathscr{F})$

Finally, Goodman also defined a model stronger than *cache*: the *processor* [24] one. It encompasses the requirements of both *PRAM* and *cache* consistencies. Therefore:

$\textsc{ProcessorConsistency}(\mathscr{F}) \equiv \textsc{PerObjectSingleOrder} \land \text{PRAM} \land \textsc{RVal}(\mathscr{F})$

3.4 Analysis of Consistency Models

Figure 1 summarises the "*weaker than*" ($\rightarrow$, defined in Section 3.3.1) relationships among consistency models. Those relations were already identified by Viotti and Vukolić [61].

Let us take the definitions of all those consistency models as a base for analysing which of them require any kind of coordination that will be broken in case of a network partition. Those models certainly are *CAP-constrained* since they will require any kind of consensus among the participating processes. This analysis is presented in Section 3.4.1. Its results identify a larger set of *CAP-constrained* models than in other related work [50]. However, Section 3.4.2 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 3.4.3 looks for other inter-model relations that were not identified in [61].

3.4.1 Starting Point: Linearisability

Gilbert and Lynch [23] proved the CAP theorem assuming that its consistency referred to *linearisability* [26]. Let us start revising the definition of *linearisability* given in Section 3.3.1, with the goal of identifying which of its conditions cannot be respected in a partitioned and available system:

$\textsc{Linearisability}(\mathscr{F}) \equiv \textsc{SingleOrder} \land \textsc{RealTime} \land \textsc{RVal}(\mathscr{F})$

Table 1: Definition of basic consistency predicates.

Predicate	**Definition**
RVAL($\mathscr{F}$)	$\forall op \in H : op.oval \in \mathscr{F}(op, cxt(A, op))$
SINGLEORDER	$\exists H' \subseteq \{op \in H : op.oval = \nabla\} : vis = ar \setminus (H' \times H)$
REALTIME	$rb \subseteq ar$
REALTIMEWRITES	$rb \vert_{wr \rightarrow op} \subseteq ar$
SEQRVAL($\mathscr{F}$)	$\forall op \in H : Concur(op) = \emptyset \Rightarrow op.oval \in \mathscr{F}(op, cxt(A, op))$
EVENTUALVISIBILITY	$\forall a \in H, \forall [f] \in H/\approx_{ss} : \lvert \{b \in [f] : (a,b) \in rb \wedge (a,b) \notin vis\} \rvert < \infty$
NOCIRCULARCAUSALITY	$acyclic(hb)$
STRONGCONVERGENCE	$\forall a, b \in H \vert_{rd} : vis^{-1}(a) \vert_{wr} = vis^{-1}(b) \vert_{wr} \Rightarrow a.oval = b.oval$
CAUSALVISIBILITY	$hb \subseteq vis$
CAUSALARBITRATION	$hb \subseteq ar$
TIMEDVISIBILITY(Δ)	$\forall a \in H \vert_{wr}, \forall b \in H, \forall t \in Time : a.rtime = t \wedge b.stime = t + \Delta$
	$\Rightarrow (a,b) \in vis$
REALTIMEWW	$rb \vert_{wr \rightarrow wr} \subseteq ar$
K-REALTIMEREADS(K)	$\forall a \in H \vert_{wr}, \forall b \in H \vert_{rd}, \forall PW \subseteq H \vert_{wr}, \forall pw \in PW : \lvert PW \rvert < K \wedge$
	$(a, pw) \in ar \wedge (pw, b) \in rb \wedge (a,b) \in rb \Rightarrow (a,b) \in ar$
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	$\forall a_i, a_j \in H : a_i \not\approx_{ss} a_j \wedge (a_i, a_j) \in ar \setminus vis \Rightarrow \lvert \{b_i \in H : a_i \preceq_{so} b_i \wedge$
	$(\exists b_j \in H : a_j \preceq_{so} b_j \wedge (b_i, b_j) \in vis)\} \rvert \leq 1 \wedge \lvert \{b_j \in H : a_j \preceq_{so} b_j$
	$\wedge (\exists b_i \in H : a_i \preceq_{so} b_i \wedge (b_j, b_i) \in vis)\} \rvert \leq 1$
PEROBJECTSINGLEORDER	$\exists H' \subseteq \{op \in H : op.oval = \nabla\} : vis \cap ob = ar \cap ob \setminus (H' \times H)$

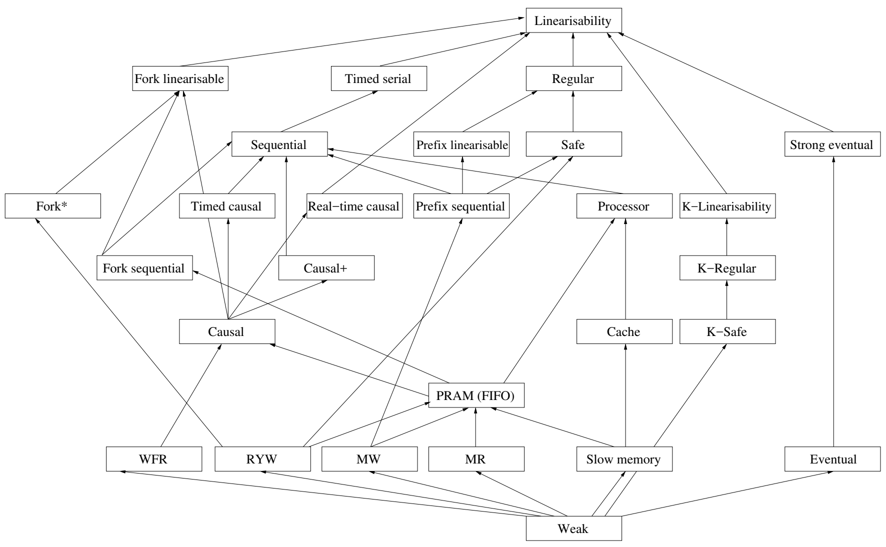

Figure 1: Strength-based partial ordering of consistency models.

RVAL($\mathscr{F}$) is needed for defining the appropriateness of the return value in an operation call based on its context found in that execution. The other two predicates mean the following (Table 1):

- SINGLEORDER: $\exists H' \subseteq \{op \in H : op.oval = \nabla\} : vis = ar \setminus (H' \times H)$. This predicate states that the visibility relation (*vis*) being used in the model coincides with the arbitration relation (*ar*), discarding the still incomplete operations. The arbitration relation sets a total order on the concurrent or invisible operations. As a result of this, SINGLEORDER compels every process to observe the same order of written values.

  In case of network partitions, such agreement will not be possible because of the FLP impossibility result [21]. Thus, a system $S = (P,N,O)$ is partitioned into multiple subsystems $S_i = (P_i,N_i,O)$ where $i \in K$. Besides, $\forall i, j \in K, i \neq j : Connects(P_i,N_j) = false$. This means that all write operations applied in P_i become invisible (i.e., they are not brought in their *vis* relation) for every other P_j with $i \neq j$. The goal of the *ar* relation is to bring order to those invisible operations. However, since communication between processes in P_i and P_j remains impossible while the network is partitioned, and both P_i and P_j are available to their respective clients, the $vis = ar \setminus (H' \times H)$ condition that defines SINGLEORDER cannot be ensured, since the already completed writes in a network component cannot be known by the processes belonging to other network components. Moreover, once the partition is healed, there will be no way of complying with such a condition, either: what has been arbitrated up to that point cannot be undone.

- REALTIME: $rb \subseteq ar$. This predicate requires that all the operations already ordered by the returns-before (*rb*) relation are considered by the arbitration relation (*ar*). Since *rb* is built considering real time, *rb* is able to order the operations executed by different processes, even when they are placed in different network components in case of a partition or their operations have accessed different objects.

  As in the previous case, this REALTIME predicate cannot be maintained in two different system subgroups S_1 and S_2 when the network becomes partitioned. What is being ordered in one of those subgroups according to *rb* cannot be known nor enforced in the other subgroup.

Those predicates have been used in the definition of other consistency models. According to the arguments that we have presented above, those other models will not be able to comply with the CAP constraints. Let us revise which they are:

- SINGLEORDER: *linearisability*, *regular*, *safe*, *sequential*, *prefix sequential*, *prefix linearisable*, *timed serial*, *k-linearisability*, *k-regular* and *k-safe*.

  Besides, the PEROBJECTSINGLEORDER predicate also requires consensus on a single order, but constrained per each existing object. Thus, it is more relaxed than SINGLEORDER, but it is not attainable in a partitioned system. That second predicate is needed in the definition of these models: *cache* and *processor*.

- REALTIME (or its variants REALTIMEWRITES, REALTIMEWW): *linearisability*, *regular*, *safe*, *prefix linearisable*, *k-linearisability*, *fork linearisability*, and *fork**.

Therefore, according to this first step related with the SINGLEORDER and REALTIME predicates, a preliminary borderline between CAP-constrained and CAP-free models could be set as depicted in Figure 2.

3.4.2 Exploring the Frontier

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*, *fork sequential*, *causal+*, *timed causal* and *real-time causal*. If any of these models was also CAP-constrained, we would

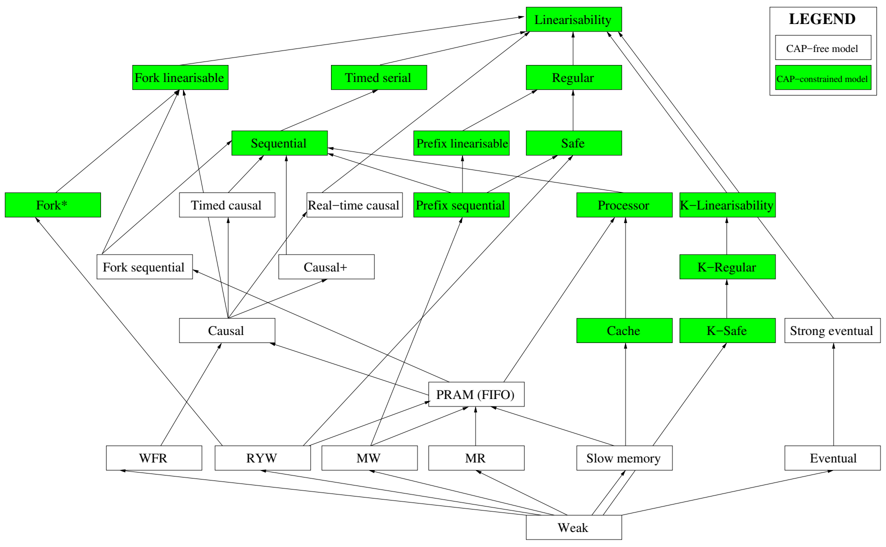

Figure 2: CAP-constrained and CAP-free consistency models considering SINGLEORDER and REALTIME.

continue our analysis studying the other 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, the other models more relaxed than those in *CCC* will not be CAP-constrained either.

The defining predicates for the currently identified *CCC* models are:

- STRONGCONVERGENCE: $\forall a,b \in H\,|_{rd}: vis^{-1}(a)\,|_{wr} = vis^{-1}(b)\,|_{wr} \Rightarrow a.oval = b.oval$. It is used in the *strong eventual* and *causal*+ models.

  STRONGCONVERGENCE requires that correct replicas that have delivered the same updates have equivalent state [54]. Its definition is based on the vis^{-1} relation and this implies that it only considers updates already delivered at the involved reader processes. Therefore, if the effects of a write cannot be delivered yet to other processes due to a network partition, STRONGCONVERGENCE does not set any requirement on those other processes. However, once network connectivity is restored, the write operations being applied need to ensure convergence. Shapiro et al [54] provide several examples of *conflict-free replicated data types* (CRDTs) that are *strongly convergent* and, therefore, are available while the network remains partitioned.

  Therefore, STRONGCONVERGENCE does not imply that the models it takes part of will be CAP-constrained.

- EVENTUALVISIBILITY: $\forall a \in H, \forall [f] \in H/\approx_{ss}: |\, \{b \in [f] : (a,b) \in rb \wedge (a,b) \notin vis\}\, | < \infty$. It is used in the *strong eventual* model.

  This property states that the amount of write events whose effects are not yet placed in the visibility relation cannot increase without limit. In other words, all write events should be eventually known by the other processes.

  Regarding the CAP constraints, EVENTUALVISIBILITY only demands that network partitions be eventually repaired. As a result, it cannot be a CAP-constrained predicate.

- NOCIRCULARCAUSALITY: $acyclic(hb)$. It is used in the *strong eventual* model.

  This property complements EVENTUALVISIBILITY in order to define *eventual* consistency. NOCIRCULARCAUSALITY restricts the eventual propagation and visibility of write operations, requiring that –once known by other processes– the effects of a write operation do not introduce any cycle in the *happens-before* relation.

  According to Burckhardt [12] this predicate was introduced in his specification of *basic eventual* consistency in order to avoid the *circular causality* [12] (also known as *thin air* [13]) anomaly that may be generated by some compiler optimisations for shared memory multiprocessors. In processes running at different computers that propagate write operations using messages, such an anomaly cannot happen.

- NOBACKWARDTIME ($\forall a, b \in H : (a,b) \in rb \Rightarrow (b,a) \notin vis$). This predicate is used in the *real-time causal* model. It precludes the operations ordered by the *returns-before* relation to be considered in opposite order in the *visibility* relation. *Visibility* requires value propagation. NOBACKWARD-TIME is easy to ensure while no network partition happens in a system, since many causally consistent systems already guarantee this [44]. On the other hand, while the network is partitioned, the *rb*-ordered operations in each network component are not known in the other components and are concurrent with the operations executed in those other components. Thus, when the network recovers connectivity, each component may forward its written values to the other components, receiving those written in them. Such value exchange must be controlled considering the constraints imposed by this NOBACKWARDTIME predicate. An accurate support of NOBACKWARDTIME for applying that exchange of object values requires precise operation timestamping. This might seem difficult to implement, but there have been systems that supported that kind of timestamping for implementing interconnectable causal communication [29, 30, 47], solving a similar problem [17]. Thus, it is feasible. Besides, a trivial implementation that does not propagate any missed written value to the other components will not violate NOBACKWARDTIME (nor CAUSALVISIBILITY or CAUSALARBITRATION, since *hb* does not include pairs where each operation has been executed in a different network component). Therefore, NOBACKWARDTIME is a CAP-free predicate, since it can be easily respected in a partitionable system.

- CAUSALVISIBILITY ($hb \subseteq vis$) and CAUSALARBITRATION ($hb \subseteq ar$). These properties are used in the *causal+*, *real-time causal* and *timed causal* models and both compose the definition of the *causal* consistency model.

  The *happens-before* (*hb*) relation is based on local execution order or on the order of write propagation events (i.e., $write \rightarrow read$) between processes. CAUSALVISIBILITY and CAUSALARBITRATION state, respectively, that the partial order being set by *hb* is maintained in the *vis* and *ar* relations. If a network partition divides the system network, write propagations cannot occur between disjoint network components. Therefore, all write events happening at disjoint components will be considered *concurrent* according to the *hb* relation and they will be considered correct and accepted in the resulting history. Because of this, both predicates can be used without problems in CAP-free models.

- TIMEDVISIBILITY(Δ): $\forall a \in H\mid_{wr}, \forall b \in H, \forall t \in Time : a.rtime = t \wedge b.stime = t + \Delta \Rightarrow (a,b) \in vis$. It is used in the *timed causal* model.

  The TIMEDVISIBILITY(Δ) predicate requires that if a write operation a is executed by a process at time t, that written value will be visible at all other processes at time $t+\Delta$.

  In case of partitions, what has been written in a network component cannot be transmitted to the remaining network components. TIMEDVISIBILITY(Δ) will not be respected if the network remains partitioned for an interval longer than Δ.

  Therefore, TIMEDVISIBILITY(Δ) is a CAP-constrained predicate when network partitions last longer than Δ. On the other hand, it could be CAP-free with $\Delta = \infty$, since in that case the *timed causal* model becomes equivalent to the *causal* one.

- PRAM: $so \subseteq vis$. It is needed in the *fork sequential* model.

  The PRAM predicate states that the *session order* (so) relation is maintained by the *visibility* (vis) relation. This means that the values written by each process should be considered by all the remaining processes in that writing order. However, no constraint is placed on the mixing of writes generated by different processes. Therefore, *so* only demands that the local writing order will be eventually known by the other processes. There is no timeliness requirement in PRAM. As a result of this, network partitions do not prevent PRAM from being respected. Once the partition failure is healed, each writer may propagate its writes in FIFO order to the processes contained in the other network components. Thus, PRAM is a CAP-free predicate.

- NOJOIN: $\forall a_i, b_i, a_j, b_j \in H : a_i \not\approx_{ss} a_j \wedge (a_i, a_j) \in ar \setminus vis \wedge a_i \preceq_{so} b_i \wedge a_j \preceq_{so} b_j \Rightarrow (b_i, b_j), (b_j, b_i) \notin vis$. It is needed in the *fork sequential* model.

  The NOJOIN predicate provides the key characteristic of the *fork* models: once the execution of two processes has become forked because of an arbitrary failure, their forked executions will not be joined (i.e., merged) again.

  Thus, NOJOIN is a divergence predicate and its compliance will not be impeded by network partitions. Therefore, it is a CAP-free predicate.

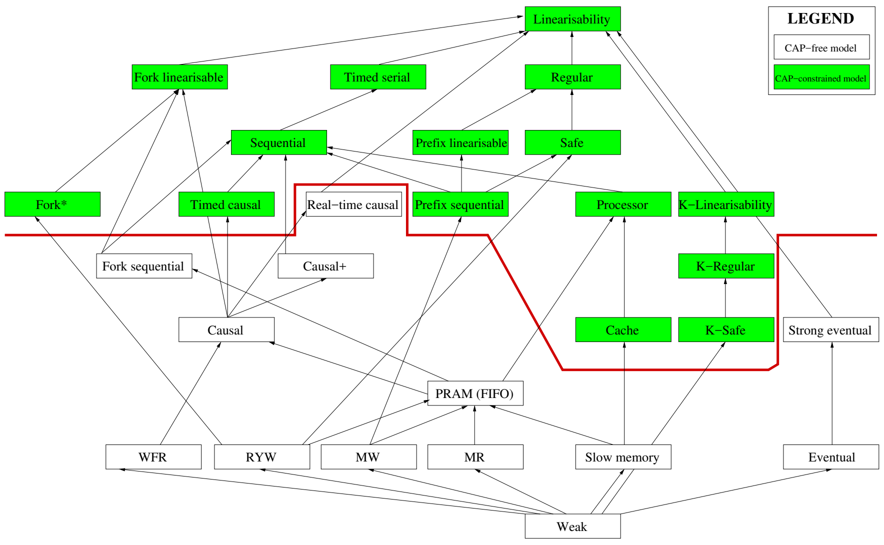

Figure 3: CAP-constrained vs CAP-free borderline.

This analysis has shown that the TIMEDVISIBILITY predicate forces its user models to be CAP-constrained. As a result of this, the *timed causal* model is also CAP-constrained. Figure 2 shows that *timed causal* is directly stronger than *causal* and there is no other more relaxed model directly related with *timed causal*. The *causal* model consists of the CAUSALVISIBILITY and CAUSALARBITRATION predicates. Both have been analysed above, showing that they are CAP-free. Due to this, no other CAP-constrained model may be found and this analysis should end here. Thus, the borderline between CAP-constrained and CAP-free models is finally set as depicted in Figure 3.

3.4.3 Other Inter-Model Relations

Although Figure 2 shows some "weaker than" relations between models according to what was presented in [61], there are some others that have not been represented yet in that diagram. The characterisation of distributed consistency models given in [61] and those presented by the proposers of each model may be used to depict several missing relationships, as shown in Figure 4 and explained hereafter.

Indeed, Viotti and Vukolić [61] show others in their paper, without formally deriving them from the predicates that define each model. For instance, they include the *fork-join causal* [43] (FJC) and *bounded fork-join causal* (BFJC) [44] models, but no formal specification is given for them. FJC maintains causal consistency among correct processes, and inconsistent writes generated by a Byzantine process are considered as concurrent writes generated by virtual processes. Viotti and Vukolić [61] justify that FJC consistency is weaker than *causal* and *BFJC* is weaker than *fork* and stronger than FJC. Additionally, Mahajan et al [44] prove that BFJC is CAP-free and this implies that FJC is CAP-free, too.

Another example of this kind affects the family of *probabilistic bounded staleness* (PBS) [7] consistency models proposed by Bailis et al. It 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

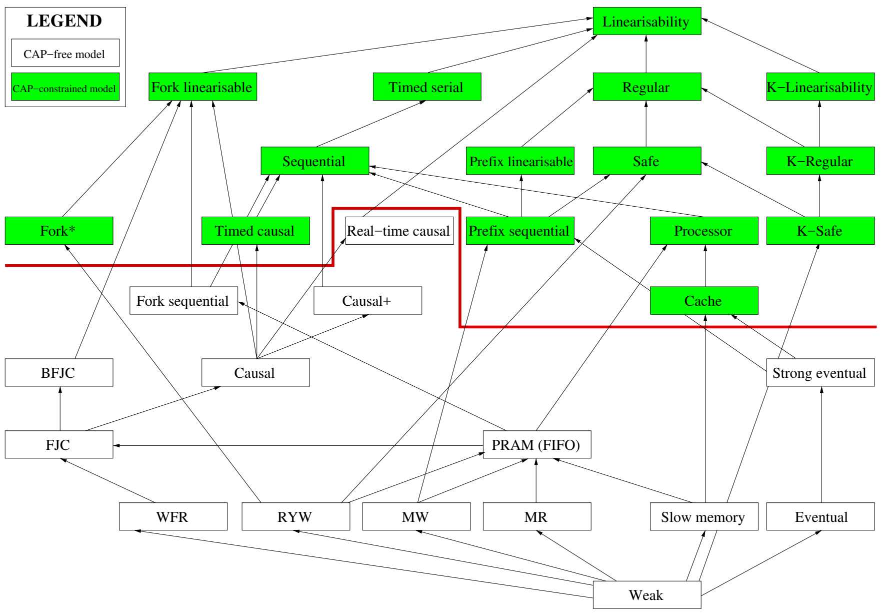

Figure 4: Revised "*weaker than*" ($\rightarrow$) relationships among models.

the time needed by written values to become visible. The third one combines the other two. Viotti and Vukolić [61] state that PBS k-staleness, depending on its parameter k, may be weaker than or equivalent to *k-linearisability*, while PBS t-visibility is a probabilistic weakening of TIMEDVISIBILITY(Δ). 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-linear and, as such, is CAP-constrained.

As Brewer states in [11] "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, able to 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 [4, 34, 20, 63, 33, 52, 32, 53, 39, 14, 19, 9, 8, 25] of this kind, that are analysed by Viotti and Vukolić [61] in its eighth group: *composable and tunable* models. We refer the reader to [61] for a short description and comparison of them. Many of those models admit configurations in both parts of our identified frontier.

Eventual consistency was defined with the goal of breaking the constraints of the CAP theorem [6]. Thus, it is one of the best known representatives of the CAP-free set of models. It is interesting to relate it with CAP-constrained models. Intuitively, it may be weaker than all of them. Indeed, even *strong eventual* [54] consistency seems to be weaker than many of them. Let us consider that relationship.

Section 3.4.1 showed that SINGLEORDER sets an agreement on the order of writes for all participating processes. At a glance, that condition satisfies the requirements of the STRONGCONVERGENCE predicate. The latter demands that once the same set of write operations is applied by two processes, their states converge [54] and any reads in those processes on the same object return the same value. SINGLEORDER guarantees that such set of writes is seen in the same order by those processes. Therefore, all executions respecting SINGLEORDER comply with STRONGCONVERGENCE and, additionally, they also trivially comply with EVENTUALVISIBILITY and NOCIRCULARCAUSALITY, since SINGLEORDER requires that all completed writes are visible to all participating processes (this implies EVENTUALVISIBILITY) and its agreed order avoids cycles in the *hb* relation (i.e., NOCIRCULARCAUSALITY). Thus, executions complying with SINGLEORDER are *strongly eventual*. On the contrary, there are some *strongly eventual* executions that do not respect SINGLEORDER [54]; e.g., those based on commutative write operations that are applied in different order in two different processes: they converge without following the same write-order in every process. With this, it can be stated that every model based on SINGLEORDER is strictly stronger than *strong eventual* consistency.

Indeed, what has been said about SINGLEORDER also applies to PEROBJECTSINGLEORDER. Therefore, the *strong eventual* model is weaker than *prefix sequential* and *cache*. *K-safe*, *K-regular* and *K-linearisable* consistencies respect SINGLEORDER but they also extend the set of values that may be returned by read operations, relaxing the resulting consistency, as shown in Figure 4. Although every replica applies the same sequence of writes and their states converge, subsequent read operations may return any of the most K-recent values. Because of this, there is no guarantee that reads executed on different replicas that have applied the same set of write operations return the same value (i.e., STRONGCONVERGENCE is not respected). Thus, *K-safe*, *K-regular* and *K-linearisable* consistencies are incomparable with *strong eventual* consistency.

Besides, the *timed causal* model is also incomparable with the *strong eventual* model. Let us consider two executions in order to show this. Let us start with a *timed causal* execution. It consists of two write operations assigning different values to the same object that occur at the same real-time instant in two different processes p_1 and p_2. Such writes are locally observable in both processes at that time and need some interval (shorter than Δ) for being transmitted to the other process. When they are delivered, they are accepted and read by the other process. As a result of this, both p_1 and p_2 have received the same set of values on the same object, but their respective reads have returned different values at the end of this execution. Therefore, this execution does not respect STRONGCONVERGENCE and this example means that *timed causal* seems to be weaker than *strong eventual*.

But we may also find a *strongly eventual* execution that is not *timed causal*. The system takes $\Delta = 2$ in order to manage *timed causal* consistency. Let us consider an execution consisting of two write operations on the same object from p_1 and p_2. Process p_1 writes value v_1 and one time unit later p_2 receives and reads that value before writing value v_2. Once this is made, the network is partitioned and remains partitioned 3 time units. Later on, connectivity is resumed, and value v_2 is propagated and delivered to p_1. Thus, now both processes know about the values written by the other and both agree that the final value is v_2, complying with the STRONGCONVERGENCE condition. However, in this case, TIMEDVISIBILITY(2) has not been held due to the temporary network partition. Then, this execution is not *timed causal* but it is *strong eventual*. Therefore, both models are incomparable.

Recently, Attiya et al [6] have formally proved that *observable causal* consistency (a model slightly stronger than *causal*) is the strongest eventual (i.e., CAP-free and non-inherently convergent) model that can be supported by either multi-value registers or observed-remove sets in a partitionable system, assuming that read operations do not modify the state and messages are generated only following an operation. It is incomparable with the *real-time causal* model [6] since both take different mechanisms to reach convergence. That result confirms the validity of our identified frontier between CAP-constrained and CAP-free models, at least on what regards to placing *causal* consistency in the strongest subset of the CAP-free domain.

4 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 [42, 44, 6]. Those papers have taken as a base causal consistency, adding some conditions in order to strengthen it, generating in that way the *causal+* [42], *real-time causal* [44] and *observable causal* [6] models. Those models are incomparable to each other and they define the strongest subset of models in the CAP-free set.

Another "classical" approach to implement available and partition-tolerant services is based on *eventual* consistency. The term *eventual consistency* was probably first used in the Clearinghouse system [18] and in the Lotus/Iris Notes CSCW (computer-supported cooperative work) project [31], 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 [28, 15] (1975) and Alsberg and Day [3] (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 [21]) or a close to immediate recency on the values read (broken when the latest values have been written in another network component while the network is partitioned). The consistency specification framework proposed by Burckhardt et al [13, 12] provides an excellent basis for specifying consistency models. With it, it is easy to specify both safety and liveness conditions. Viotti and Vukolić [61] have already used that framework for surveying distributed consistency models. Our work complements their survey looking for the CAP-constrained to CAP-free frontier considering value-order consensus and read recency criteria. Thus, the current classification extends other previous work focused on setting a partial frontier based on value-order consensus [50, 49].

A complete frontier is not easy to set. Indeed, Viotti and Vukolić show some of the existing *weaker-than* relations among models and, regarding the *eventual* and *strong eventual* models, they only state that *strong eventual* is weaker than *linearisability*. However, we have shown that *strong eventual* is weaker than *cache* consistency (and all other models stronger than cache), although it is not weaker than all CAP-constrained models; e.g., it is incomparable with *fork**, *timed causal* and *K-safe*, and these three are CAP-constrained.

There are a good set of composable and tunable consistency models [4, 34, 20, 63, 33, 52, 32, 53, 7, 39, 14, 19, 9, 8, 25] whose implementation protocols simultaneously support both CAP-constrained and CAP-free consistencies. Thus, that seems to be the best approach to overcome the limitations imposed by the CAP theorem on the consistency of highly available distributed services since they may relax consistency while the network remains partitioned or easily determine which service activities may remain blocked or partially inconsistent while partitions arise. Additionally, they support and provide quite a strong consistency while the network remains connected.

5 Conclusions

We have revised which distributed consistency models, besides *linearisability*, are directly affected by the CAP theorem constraints on consistency. This study has shown that there are many other models (e.g., fork linearisable, timed serial, regular, sequential, prefix linearisable, safe, fork*, timed causal, processor, cache, k-linearisable,...) that cannot be ensured in available and partition-tolerant distributed services. Therefore, the strongest classical data-centric consistency to be supported by those services in case of network partitions is the causal one that may be complemented with other conditions in order to provide stronger semantics in a partitionable system, building in this way the *causal+*, *real-time causal* and *observable causal* models. Many other CAP-free models have been also identified. Thus, a more precise frontier between CAP-constrained and CAP-free models has been set in this paper, providing a good basis for classifying any newly proposed models from now on.

References

[1] Ahamad M, Burns JE, Hutto PW, Neiger G (1991) Causal memory. In: 5th International Workshop on Distributed Algorithms and Graphs (WDAG), Delphi, Greece, pp 9–30, DOI 10.1007/BFb0022435

[2] Aiyer AS, Alvisi L, Bazzi RA (2005) On the availability of non-strict quorum systems. In: 19th International Conference on Distributed Computing (DISC), Cracow, Poland, pp 48–62, DOI 10.1007/11561927_6

[3] Alsberg P, Day JD (1976) A principle for resilient sharing of distributed resources. In: 2nd International Conference on Software Engineering (ICSE), San Francisco, CA, USA, pp 562–570

[4] Attiya H, Friedman R (1992) A correctness condition for high-performance multiprocessors. In: 24th Annual ACM Symposium on Theory of Computing (STOC), Victoria, British Columbia, Canada, pp 679–690, DOI 10.1145/129712.129778

[5] Attiya H, Friedman R (1996) Limitations of fast consistency conditions for distributed shared memories. Inf Process Lett 57(5):243–248, DOI 10.1016/0020-0190(96)00007-5

[6] Attiya H, Ellen F, Morrison A (2017) Limitations of highly-available eventually-consistent data stores. IEEE Trans Parallel Distrib Syst 28(1):141–155, DOI 10.1109/TPDS.2016.2556669

[7] Bailis P, Venkataraman S, Franklin MJ, Hellerstein JM, Stoica I (2012) Probabilistically bounded staleness for practical partial quorums. PVLDB 5(8):776–787

[8] Balegas V, Duarte S, Ferreira C, Rodrigues R, Preguiça NM, Najafzadeh M, Shapiro M (2015) Putting consistency back into eventual consistency. In: 10th European Conference on Computer Systems (EuroSys), Bordeaux, France, pp 6:1–6:16, DOI 10.1145/2741948.2741972

[9] Bessani AN, Mendes R, Oliveira T, Neves NF, Correia M, Pasin M, Veríssimo P (2014) SCFS: A shared cloud-backed file system. In: USENIX Annual Technical Conference (ATC), Philadelphia, PA, USA, pp 169–180

[10] Birman KP, Friedman R (1996) Trading consistency for availability in distributed systems. Tech. rep., 96-1579, Dept. of Comput. Sc., Cornell University, Ithaca, NY, USA

[11] Brewer EA (2012) CAP twelve years later: How the "rules" have changed. IEEE Computer 45(2):23–29, DOI 10.1109/MC.2012.37

[12] Burckhardt S (2014) Principles of eventual consistency. Foundations and Trends in Programming Languages 1(1-2):1–150, DOI 10.1561/2500000011

[13] Burckhardt S, Gotsman A, Yang H, Zawirski M (2014) Replicated data types: specification, verification, optimality. In: 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL), San Diego, CA, USA, pp 271–284, DOI 10.1145/2535838.2535848

[14] Chihoub H, Ibrahim S, Antoniu G, Pérez-Hernández MS (2012) Harmony: Towards automated self-adaptive consistency in cloud storage. In: IEEE International Conference on Cluster Computing (CLUSTER), Beijing, China, pp 293–301, DOI 10.1109/CLUSTER.2012.56

[15] Cosell BS, Johnson PR, Malman JH, Schantz RE, Sussman J, Thomas RH, Walden DC (1975) An operational system for computer resource sharing. In: 5th ACM Symposium on Operating System Principles (SOSP), The University of Texas at Austin, Austin, Texas, USA, pp 75–81

[16] Davidson SB, García-Molina H, Skeen D (1985) Consistency in partitioned networks. ACM Comput Surv 17(3):341–370, DOI 10.1145/5505.5508

[17] de Juan-Marín R, Decker H, Armendáriz-Íñigo JE, Bernabéu-Aubán JM, Muñoz-Escoí FD (2016) Scalability approaches for causal multicast: A survey. Computing 98(9):923–947, DOI 10.1007/s00607-015-0479-0

[18] Demers AJ, Greene DH, Hauser C, Irish W, Larson J, Shenker S, Sturgis HE, Swinehart DC, Terry DB (1987) Epidemic algorithms for replicated database maintenance. In: 6th ACM Symposium on Principles of Distributed Computing (PODC), Vancouver, British Columbia, Canada, pp 1–12, DOI 10.1145/41840.41841

[19] Dobre D, Viotti P, Vukolić M (2014) Hybris: Robust hybrid cloud storage. In: ACM Symposium on Cloud Computing (SoCC), Seattle, WA, USA, pp 12:1–12:14, DOI 10.1145/2670979.2670991

[20] Fekete A, Gupta D, Luchangco V, Lynch NA, Shvartsman AA (1996) Eventually-serializable data services. In: 15th ACM Symposium on Principles of Distributed Computing (PODC), Philadelphia, PA, USA, pp 300–309

[21] Fischer MJ, Lynch NA, Paterson M (1985) Impossibility of distributed consensus with one faulty process. J ACM 32(2):374–382, DOI 10.1145/3149.214121

[22] Fox A, Brewer EA (1999) Harvest, yield and scalable tolerant systems. In: 7th Workshop on Hot Topics in Operating Systems (HotOS), Rio Rico, Arizona, USA, pp 174–178, DOI 10.1109/HOTOS.1999.798396

[23] Gilbert S, Lynch N (2002) Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News 33(2):51–59, DOI 10.1145/564585.564601

[24] Goodman JR (1989) Cache consistency and sequential consistency. Tech. rep., Number 61, IEEE Scalable Coherent Interface Working Group

[25] Gotsman A, Yang H, Ferreira C, Najafzadeh M, Shapiro M (2016) 'cause I'm strong enough: reasoning about consistency choices in distributed systems. In: 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL), St. Petersburg, FL, USA, pp 371–384, DOI 10.1145/2837614.2837625

[26] Herlihy M, Wing JM (1990) Linearizability: A correctness condition for concurrent objects. ACM Trans Program Lang Syst 12(3):463–492, DOI 10.1145/78969.78972

[27] Hutto PW, Ahamad M (1990) Slow memory: Weakening consistency to enchance concurrency in distributed shared memories. In: 10th International Conference on Distributed Computing Systems (ICDCS), Paris,France, pp 302–309, DOI 10.1109/ICDCS.1990.89297

[28] Johnson PR, Thomas RH (1975) The maintenance of duplicate databases. RFC 677, Network Working Group, Internet Engineering Task Force

[29] Kawanami S, Enokido T, Takizawa M (2004) A group communication protocol for scalable causal ordering. In: 18th International Conference on Advanced Information Networking and Applications (AINA), Fukuoka, Japan, pp 296–302

[30] Kawanami S, Nishimura T, Enokido T, Takizawa M (2005) A scalable group communication protocol with global clock. In: 19th International Conference on Advanced Information Networking and Applications (AINA), Taipei, Taiwan, pp 625–630, DOI 10.1109/AINA.2005.58

[31] Kawell L Jr, Beckhardt S, Halvorsen T, Ozzie R, Greif I (1988) Replicated document management in a group communication system. In: ACM Conference on Computer-Supported Cooperative Work (CSCW), Portland, Oregon, USA, pp 395–, DOI 10.1145/62266.1024798

[32] Kraska T, Hentschel M, Alonso G, Kossmann D (2009) Consistency rationing in the cloud: Pay only when it matters. PVLDB 2(1):253–264

[33] Krishnamurthy S, Sanders WH, Cukier M (2002) An adaptive framework for tunable consistency and timeliness using replication. In: International Conference on Dependable Systems and Networks (DSN), Bethesda, MD, USA, pp 17–26, DOI 10.1109/DSN.2002.1028882

[34] Ladin R, Liskov B, Shrira L, Ghemawat S (1992) Providing high availability using lazy replication. ACM T Comput Syst 10(4):360–391, DOI 10.1145/138873.138877

[35] Lamport L (1978) Time, clocks, and the ordering of events in a distributed system. Commun ACM 21(7):558–565, DOI 10.1145/359545.359563

[36] Lamport L (1979) How to make a multiprocessor computer that correctly executes multiprocess programs. IEEE T Comput 28(9):690–691, DOI 10.1109/TC.1979.1675439

[37] Lamport L (1986) On interprocess communication. Part II: algorithms. Distrib Comput 1(2):86–101, DOI 10.1007/BF01786228

[38] Lamport L, Shostak RE, Pease MC (1982) The byzantine generals problem. ACM Trans Program Lang Syst 4(3):382–401, DOI 10.1145/357172.357176

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

[40] Li J, Mazières D (2007) Beyond one-third faulty replicas in byzantine fault tolerant systems. In: 4th Symposium on Networked Systems Design and Implementation (NSDI), Cambridge, Massachusetts, USA, pp 131–144

[41] Lipton RJ, Sandberg JS (1988) PRAM: A scalable shared memory. Tech. rep., CS-TR-180-88, Princeton University, USA

[42] Lloyd W, Freedman MJ, Kaminsky M, Andersen DG (2011) Don't settle for eventual: scalable causal consistency for wide-area storage with COPS. In: 23rd ACM Symposium on Operating Systems Principles (SOSP), Cascais, Portugal, pp 401–416, DOI 10.1145/2043556.2043593

[43] Mahajan P, Setty STV, Lee S, Clement A, Alvisi L, Dahlin M, Walfish M (2010) Depot: Cloud storage with minimal trust. In: 9th USENIX Symposium on Operating Systems Design and Implementation, (OSDI), Vancouver, BC, Canada, pp 307–322

[44] Mahajan P, Alvisi L, Dahlin M (2011) Consistency, availability and covergence. Tech. rep., UTCS TR-11-22, Dept. of Computer Science, The University of Texas at Austin, USA

[45] Mazières D, Shasha DE (2002) Building secure file systems out of byantine storage. In: 21st Annual ACM Symposium on Principles of Distributed Computing (PODC), Monterrey, California, USA, pp 108–117, DOI 10.1145/571825.571840

[46] Muñoz-Escoí FD, Bernabéu-Aubán JM (2017) A survey on elasticity management in PaaS systems. Computing 99(7):617–656, DOI 10.1007/s00607-016-0507-8

[47] Nishimura T, Hayashibara N, Takizawa M, Enokido T (2005) Causally ordered delivery with global clock in hierarchical group. In: 14th International Conference on Parallel and Distributed Systems (ICPADS), Fukuoka, Japan, pp 560–564, DOI 10.1109/ICPADS.2005.105

[48] Oprea A, Reiter MK (2006) On consistency of encrypted files. In: 20th International Symposium on Distributed Computing (DISC), Stockholm, Sweden, pp 254–268, DOI 10.1007/11864219_18

[49] Pascual-Miret L, Muñoz-Escoí FD (2016) Replica divergence in data-centric consistency models. In: 27th International Workshop on Database and Expert Systems Applications (DEXA Workshops), Porto, Portugal, pp 109–112, DOI 10.1109/DEXA.2016.035

[50] Pascual-Miret L, González de Mendívil JR, Bernabéu-Aubán JM, Muñoz-Escoí FD (2015) Widening CAP consistency. Tech. rep., IUMTI-SIDI-2015/03, Inst. Univ. Mixto Tecnológico de Informática, Univ. Politècnica de València, Spain

[51] Saito Y, Shapiro M (2005) Optimistic replication. ACM Comput Surv 37(1):42–81, DOI 10.1145/1057977.1057980

[52] Santos N, Veiga L, Ferreira P (2007) Vector-field consistency for ad-hoc gaming. In: ACM/IFIP/USENIX 8th International Middleware Conference, Newport Beach, CA, USA, pp 80–100, DOI 10.1007/978-3-540-76778-7_5

[53] Serafini M, Dobre D, Majuntke M, Bokor P, Suri N (2010) Eventually linearizable shared objects. In: 29th Annual ACM Symposium on Principles of Distributed Computing (PODC), Zurich, Switzerland, pp 95–104, DOI 10.1145/1835698.1835723

[54] Shapiro M, Preguiça NM, Baquero C, Zawirski M (2011) Conflict-free replicated data types. In: 13th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS), Grenoble, France, pp 386–400, DOI 10.1007/978-3-642-24550-3_29

[55] Tanenbaum AS, van Steen M (2007) Distributed Systems - Principles and Paradigms, 2nd Edition. Pearson Education, ISBN: 978-0-13-239227-3

[56] Terry DB, Demers AJ, Petersen K, Spreitzer M, Theimer M, Welch BB (1994) Session guarantees for weakly consistent replicated data. In: 3rd International Conference on Parallel and Distributed Information Systems (PDIS), Austin, Texas, USA, pp 140–149, DOI 10.1109/PDIS.1994.331722

[57] Terry DB, Theimer M, Petersen K, Demers AJ, Spreitzer M, Hauser C (1995) Managing update conflicts in Bayou, a weakly connected replicated storage system. In: 15th ACM Symposium on Operating System Principles (SOSP), Copper Mountain Resort, Colorado, USA, pp 172–183, DOI 10.1145/224056.224070

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

[59] Torres-Rojas FJ, Ahamad M, Raynal M (1999) Timed consistency for shared distributed objects. In: 18th Annual ACM Symposium on Principles of Distributed Computing (PODC), Atlanta, Georgia, USA, pp 163–172, DOI 10.1145/301308.301350

[60] Torres-Rojas FJ, Ahamad M, Raynal M (2002) Real-time based strong consistency for distributed objects. Comput Syst Sci Eng 17(2):133–142

[61] Viotti P, Vukolić M (2016) Consistency in non-transactional distributed storage systems. ACM Comput Surv 49(1):19:1–19:34, DOI 10.1145/2926965

[62] Vogels W (2009) Eventually consistent. Commun ACM 52(1):40–44, DOI 10.1145/1435417.1435432

[63] Yu H, Vahdat A (2002) Design and evaluation of a conit-based continuous consistency model for replicated services. ACM Trans Comput Syst 20(3):239–282