Widening CAP Consistency

Leticia Pascual-Miret, José Ramón González de Mendívil,
José M. Bernabéu-Aubán, Francesc D. Muñoz-Escoí

Instituto Universitario Mixto Tecnológico de Informática
Universitat Politècnica de València
46022 Valencia (SPAIN)

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

Technical Report IUMTI-SIDI-2015/003

Widening CAP Consistency

Leticia Pascual-Miret, José Ramón González de Mendívil,
José M. Bernabéu-Aubán, Francesc D. Muñoz-Escoí

Instituto Universitario Mixto Tecnológico de Informática
Universitat Politècnica de València
46022 Valencia (SPAIN)

Technical Report IUMTI-SIDI-2015/003

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

Revision: May 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. The correctness of this theorem has been 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 cannot be supported in an available and partition-tolerant service. Other weaker consistency models could be maintained when scalable services are deployed in partitionable systems. According to our results, part of this consistency borderline is set between the cache (unsupported) and causal+ (supported) models.

Keywords: Inter-replica consistency, CAP theorem, Service availability, Network partition.

1 Introduction

Scalable distributed services try to maintain their service continuity in all situations. When they are geo-replicated (i.e., spread across multiple data centers) 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 had been suggested long time ago [11] (Davidson et al., 1985), thoroughly explained in [15] (Fox and Brewer, 1999) and proved in [16] (Gilbert and Lynch, 2002). However, the compromise between strongly consistent actions, availability and tolerance to network partitions was already implicit in [20] (Johnson and Thomas, 1975) and justified by Birman and Friedman in [7] (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 [23]; i.e., that all instances are able to maintain the same values for each variable at the same time, providing a behavior equivalent to that of a single copy. Therefore, it is not surprising that Gilbert and Lynch assumed that kind of consistency in their proof of the CAP theorem [16].

With the advent of cloud computing, it is easy to develop and deploy highly scalable distributed services [26]. Those applications usually provide world-wide services: they are deployed in multiple data centers and this implies that network partition tolerance is a must for those services. Once partition tolerance is assumed, those services regularly prioritize availability when they should deal with the constraints of the CAP theorem. As a result, consistency is the regular 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 be still quite strong most of the time, relaxing it when any 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 models are not supported when the network becomes partitioned. We call those models *CAP-strong*. Two questions arise in this scope: (1) Does CAP affect only to atomic consistency or are there any other "CAP-strong" models? (2) If there were any other models, where could we set the border between CAP-strong and *CAP-relaxed* models (i.e., those that may be supported while both availability and network partition tolerance are guaranteed)? Let us explain the answer to these questions in the following sections, proving the correctness of these responses in the appendix.

2 System Model

We assume a partially synchronous distributed system where processes may fail by crashing. These processes run in multiple computers. Scalable distributed services may be deployed in that system. Those distributed services consist of multiple elements. Some of those elements may be replicated in order to improve their performance or their availability. In those cases, part of their state is replicated using a replication protocol and respecting some replica consistency model.

The interconnecting network might fail generating a temporary network partition. In a partitioned network multiple disjoint network components exist [9]; i.e., 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 answered in [7], since the proof given by Birman and Friedman 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 actions. This is the main requirement: considering all replicated variables managed by a set of servers, all replicas should agree on a given sequence of actions. Moreover, such sequence of actions, 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* [22]. Therefore, considering [7] there are at least two different CAP-strong models: atomic and sequential.

Birman and Friedman [7] gave another hint related to CAP: one way to relax consistency in order to maintain availability in partitioned systems is to rely on commutative actions. Commutative actions allow temporary divergences among isolated components. Once the network partition disappears and connectivity is resumed, those actions 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 [13], 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) in pages 568-569 of [3].

Commutative actions still provide the key to answer our second question. In order to break the CAP constraints we need some kinds of actions or consistency that admit continuity even when not all processes in that system can be reached. If there were any consistency models allowing such behavior, they would be CAP-relaxed. A hint to answer this question was provided by Attiya and Friedman in [4]. They identified "*fast*" consistency models: those that may complete both read and write actions faster than the network delay; i.e., without contacting other processes. This means that the updates caused by a write action will be known by the other replicas once the write has already returned control to its invoker. Intuitively, all non-fast models (e.g., atomic, sequential, and some interpretations of the processor model) are strong enough to be qualified as CAP-strong. However, cache consistency is fast (since there is at least one algorithm that supports that model in a fast way [19]) and it cannot be maintained in two disjoint network components providing a global cache-consistent image. So, fast models are a good step in the right direction, although a weaker condition is still needed for characterizing CAP-relaxed models.

What we need is a condition that does not demand per-action convergence. It should admit that every process goes on even when the network is partitioned. These models may be implemented with lazy write propagation and they should provide a global image that complies with their consistency model even when the network is partitioned. This additional requirement on consistency conditions may be named "*partitionable*" and is defined as follows:

Definition 1. *A consistency model is partitionable when its consistency requirements are still met when writes being generated by processes may take an arbitrary long time to be propagated to (a subset of) the other system processes.*

Note that such an "*arbitrary long time*" is caused by network partitions. It will become shorter once the network partition is repaired and all processes that belong to that system recover their connectivity.

If we centre our discussion in data-centric consistency models, Steinke and Nutt [30] state a few consistency properties that are able to specify the regular models of this kind. Those properties are formally specified in Appendix A.1 and they can be summarized as follows:

GPO (Global Process Order). There is a global agreement on the order of writes at each processor. Writes from different processors may be freely interleaved by each reader.

GDO (Global Data Order). There is global agreement on the order of writes on each variable.

GWO (Global Write-read-write Order). There is a global agreement on the order of potentially causal-related writes; i.e., write A globally precedes write B when the value written in A had been read by process p before it wrote B.

GAO (Global Anti Order). There is a global agreement on the order of any two writes when a process can prove that it read one before the other.

Steinke and Nutt [30] prove that: (1) GPO defines FIFO (or PRAM) [24] consistency, (2) GDO defines cache [17] consistency, (3) GPO+GDO define a model slightly stronger than processor [17] consistency, (4) GPO+GWO define causal [1] consistency, (5) GAO is stronger than GDO, and (6) GPO+GWO+GAO define sequential [22] consistency.

Considering those facts, let us analyze which consistency properties are "fast" and "partitionable" (formal justifications are given in Appendix A.2):

- GPO is fast and partitionable. It is based on the actions executed locally by each processor. When those actions are writes, readers should see them in the order followed by their writer. No consensus is needed: the order is set by each writer independently on the others. Moreover, there is no bound on the time to propagate those writes to other processes. This prevents read actions from waiting for messages from other processes. Those actions may be locally served without any problem.

  Additionally, when a GPO-based system is partitioned, each disjoint component is able to go on globally complying with all GPO constraints, since the writes generated by different processes may be freely interleaved by readers. So, a reader may drop or delay for very long the updates generated by processes placed in unreachable components.

- GWO is also fast and partitionable. If a process starts a write action, it does not need to wait for any previous write propagation. Writes may be concurrent. Only previous writes that have been read by the writer precede the current write A. In propagation-based consistency protocols, those preceding writes to other variables can be included in the propagation message for A. This prevents both read and write actions from waiting for messages from other processes. Therefore, GWO is fast.

When a network partition arises, all writes applied in disjoint components can be considered concurrent according to the causal order that inspires the GWO property. Thus, receivers of those writes may interleave them as they prefer. Therefore, the global image being provided still complies with the GWO requirements and GWO is partitionable.

- GDO is fast but not partitionable. GDO defines a total order on the values written to each variable. Setting a total order demands agreement among all per-variable writers. Although this is hard to implement without synchronous communication in both readers and writers, some algorithms exist that avoid that communication synchrony; e.g. [19].

  In the algorithm from Jiménez et al. [19] all processes are arranged in a logical ring and a circular turn is followed to pass the write-broadcast privilege. Read and write actions return immediately their results and their control, respectively, but written values are not propagated yet. Instead, they are put in an output writeset. If two or more write actions are applied on the same variable, only the last one is maintained in that set. At a given moment, only one process S in the system has the broadcast privilege. S takes its output writeset and broadcasts it to all other processes, clearing it then. When that message is received, each other process R applies those writes onto R's variables, unless any of its local writes has been applied to the same variable. In that case, the locally written value will be placed afterwards in the global order: since that local write had already returned control to the writer and that value could have already been read, it does not make sense to accept the incoming past value. The reception of that writeset propagation message implicitly yields the broadcast privilege to the next process. With this, a global order of writes onto each variable is maintained and the algorithm is still fast for both reads and writes. The main contribution of such algorithm is that it maintains per-variable write-order global agreement relying on deferred write propagation.

  On the other hand, when a network partition arises, if the different disjoint components freely go on, there will not be any way of mixing their updates in a single global order. Because of this, GDO is not partitionable.

- GAO is not fast. Let us argue on this. GAO imposes an order on every pair of write actions. Once any of the processes in the system has seen the effects of a pair of writes in a given order, all the remaining processes are compelled to observe the same order for those two writes. Let us assume that we have two concurrent writes made by two different processes on two different variables. For instance, at time t_1, process p_1 has applied action a. Action a consists in writing value 3 on variable x. At time $t2 > t1$, but before the effects of action a are known by process p_2, p_2 applies action b. Action b consists in writing value 2 on variable y. If the consistency model being assumed were fast, both writes would return control to their respective writers before any write propagation message is sent to other processes. This means that p_1 accepts value 3 for variable x before accepting value 2 for y; i.e., in p_1 those events are ordered as $a < b$. Concurrently, p_2 accepts value 2 for variable y before accepting value 3 for variable x; i.e., in p_2 those events are ordered as $b < a$. Those two read actions on the value written by the other process violate the GAO condition. Therefore, GAO is not fast.

  Thus, algorithms like the cache variant from [19] (that allow both write and read fast actions) cannot be used for implementing GAO. Moreover, since GAO is not fast, it cannot be partitionable.

This implies that GPO+GDO (Steinke's processor consistency) and GPO+GAO+GWO (sequential) cannot be partitionable, since they are based on non-partitionable properties (GDO, GAO). With this, we answer the second question from the introduction. There are several other CAP-strong models. Besides atomic and sequential, processor (GPO+GDO) and cache (GDO) are also CAP-strong. GAO is also CAP-strong, although it does not correspond to any of the traditional data-centric consistency models. On the other hand, FIFO (GPO) and causal (GPO+GWO) are examples of CAP-relaxed models.

As a corollary, we define precisely what a CAP-strong consistency model is:

Property 1. *A consistency model is CAP-strong when there is a global agreement on the order of writes on each variable.*

The justification of that statement relies on the fact that Property 1 uses the definition of GDO [30]. This means that every reader process is able to see the same sequence of values on each variable. Therefore, Property 1 is equivalent to the following statement: *A consistency model is CAP-strong when it complies with the GDO consistency property*. This also means that: *A consistency model is CAP-relaxed when it does not comply with the GDO consistency property*. We have shown that all models based on GDO are CAP-strong.

4 Reformulating CAP

The original proofs of the CAP theorem were given in [16]. They assumed atomic consistency for the "C" property of CAP. As explained in the previous section, a wider set of consistency models can be assumed. Therefore, we propose the following theorem:

Theorem 1. *It is impossible in the partially synchronous model to implement a read / write data object that guarantees the following properties:*

- *Availability*
- *A global agreement on the order of writes to each variable*[1]

in all executions.

Proof. We prove this by contradiction. Let us assume a system with at least two available processes p_1 and p_2 that are managing a single replicated variable x with a global agreement on the order of writes. They have applied a sequence of writes α onto x until there are communication problems and those processes become placed in two disjoint components $\{C_1, C_2\}$. Since the system is still available, both processes accept client requests and apply the corresponding writes onto x. Because of this, p_1 (located in C_1) applies a new (non-empty) sequence of writes β onto x while p_2 (located in C_2) applies a different (non-empty) sequence of writes γ to x. Since those components are isolated, none of the write propagations attempted by the underlying consistency algorithm can reach the other component. Therefore, p_1 observes the $\alpha \cdot \beta$ sequence of writes onto x while p_2 observes the $\alpha \cdot \gamma$ sequence. Since $\beta \neq \gamma$, this contradicts the requirement of a global agreement on the order of writes to each variable. □

5 Consequences

Theorem 1 extends the set of consistency models where the CAP constraints apply. Instead of being only atomic consistency, several other models (linearizable, sequential, processor and cache) also participate in the trade-off among consistency, availability and network partition tolerance. Eventual consistency has been proposed as a means to break the CAP trade-off [32]. At a glance, eventual consistency seems to be a very weak consistency model. However, it is not a regular consistency model but a liveness property associated to data convergence. Indeed, the CAP-relaxed models identified in Section 3 are weaker than eventual consistency, since they are not convergent per se.

CAP-relaxed models are a good basis to manage scalable partition-tolerant services when partitions arise, since causal is the strongest model whose consistency can be guaranteed in a partitionable environment. This is a direct corollary of the CAP consistency borderline studied in Section 3, since the model that can be obtained combining the GPO and GWO properties is the causal one (GPO+GWO). This fact has been used in recent geo-replicated services proposals [2, 6, 5] in order to minimize the need of coordination among replicas, thus enhancing service scalability. Additionally, multiple old research works in the area of scalable and partitionable services had based their implementations on causal consistency with lazy propagation [13, 21]. Therefore, our paper provides a proof of correctness for what has been intuitively used in the last 30 years.

[1]This second required property is equivalent to "*GDO compliance*" or "*Cache consistency*", while the original proofs given by Gilbert and Lynch [16] required "*Atomic consistency*".

Mahajan et al. (2011) [25] also proved that causal consistency may be supported in a partitionable system, considering a different basis. However, they did not explore which consistency models define the CAP-strong set of models nor which is the most relaxed CAP-strong model in order to settle a borderline between CAP-strong and CAP-relaxed models. Note that cache and causal consistencies are not comparable, since there are causal executions that are not cache, and there are also cache executions that are not causal. So, stating that causal is the stronger model to be implemented in a partitionable environment does not imply that the most relaxed model that cannot be implemented is the cache one.

Once connectivity is recovered in a system where a network partition had happened, some strategies are needed to reach data convergence. FIFO and causal consistencies, being the models to use while the network remains partitioned, provide a basis to reach eventual consistency. Again, our identified consistency borderline is useful for justifying that some convergence proposals, like the causal+ one [2], have taken the most appropriate basis to this end.

Both FIFO and causal consistencies are interconnectable [14] and easily scalable [12]. An interconnectable consistency model is one that may be implemented using a different replication protocol in each system subgroup, providing a basis to reach a global system image that still complies with that consistency model. To this end, an interconnection protocol is needed. Cholvi et al. [10] have shown that causal and FIFO consistencies may be interconnected using FIFO update propagation. Therefore, in both cases, when a partition is healed, the updates applied in each subset may be easily merged following a FIFO propagation principle in order to recover a global causal or FIFO consistency. If stronger semantics are needed, some convergence strategies should be used in order to provide eventual consistency with either a causal or FIFO basis.

Data-centric consistency models were defined assuming a *distributed shared memory* (DSM) system. They assume that process actions are raw reads and writes to variables. CAP was specified for distributed services that use replicated data elements. Those services usually provide an interface to their clients with higher-level methods. Each one of those methods may include a long sequence of memory accesses instead of a single read or write. So, apparently, some translation is needed from the traditional DSM semantics to that of replicated objects with long operations. Fortunately, the CAP-relaxed models to be used in partition-tolerant and available services maintain their guarantees in isolated executions; i.e., when each process is temporarily isolated from the others. This means that the data updates being generated when an object method is executed can be transferred lazily once such execution is completed. So, these consistency models implicitly support the regular update spreading mechanisms used in state-propagation replication protocols[2]. Therefore, the DSM to object replication translation does not need any action in CAP-relaxed models. All CAP-strong models do not have this advantageous property. This is a direct corollary from Theorem 1.

6 Conclusions

We have delved into the CAP specification and studied which classical consistency models are directly affected by the CAP theorem. This study has shown that besides atomic consistency there are other models (e.g., sequential, processor, cache,...) that cannot be ensured in available and partition-tolerant distributed services, being cache consistency the most relaxed of those models. Therefore, the strongest consistency to be supported by those services in case of network partitions is the causal one. Besides, causal consistency may be complemented with state convergence mechanisms in order to ensure stronger semantics in a partitionable system.

Causal consistency provides useful semantics that allow the development of distributed applications; i.e., it is not a too relaxed model. However, if performance is still critical while a network partition occurs, FIFO consistency may be also considered for achieving those maximal throughput levels.

[2]There are three main types of replication protocols depending on how state updates are managed: (i) invalidation (convenient when the read/write ratio is low), (ii) state propagation (convenient when the read/write ratio is large), and (iii) operation propagation (used in state-machine replication [29], when the processing effort required by each operation is low). Most implementations of services with moderate consistency requirements [13, 18, 20, 21, 27, 28, 31] have a high read/write ratio and use state-propagation protocols.

References

[1] Mustaque Ahamad, James E. Burns, Phillip W. Hutto, and Gil Neiger. Causal memory. In *5th International Workshop on Distributed Algorithms and Graphs (WDAG)*, pages 9–30, Delphi, Greece, October 1991.

[2] Sérgio Almeida, João Leitão, and Luís E. T. Rodrigues. ChainReaction: a causal+ consistent datastore based on chain replication. In *8th Eurosys Conference*, pages 85–98, Prague, Czech Republic, April 2013.

[3] Peter Alsberg and John D. Day. A principle for resilient sharing of distributed resources. In *2nd International Conference on Software Engineering (ICSE)*, pages 562–570, San Francisco, CA, USA, October 1976.

[4] Hagit Attiya and Roy Friedman. Limitations of fast consistency conditions for distributed shared memories. *Inf. Process. Lett.*, 57(5):243–248, 1996.

[5] Peter Bailis, Alan Fekete, Michael J. Franklin, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica. Coordination avoidance in database systems. *PVLDB*, 8(3):185–196, 2014.

[6] Peter Bailis, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica. Bolt-on causal consistency. In *ACM International Conference on Management of Data (SIGMOD)*, pages 761–772, New York, NY, USA, June 2013.

[7] Kenneth P. Birman and Roy Friedman. Trading consistency for availability in distributed systems. Technical report, 96-1579, Dept. of Comput. Sc., Cornell University, Ithaca, NY, USA, April 1996.

[8] Eric A. Brewer. CAP twelve years later: How the "rules" have changed. *IEEE Computer*, 45(2):23–29, February 2012.

[9] Gregory Chockler, Idit Keidar, and Roman Vitenberg. Group communication specifications: a comprehensive study. *ACM Comput. Surv.*, 33(4):427–469, 2001.

[10] Vicent Cholvi, Ernesto Jiménez, and Antonio Fernández Anta. Interconnection of distributed memory models. *J. Parallel Distrib. Comput.*, 69(3):295–306, 2009.

[11] Susan B. Davidson, Héctor García-Molina, and Dale Skeen. Consistency in partitioned networks. *ACM Comput. Surv.*, 17(3):341–370, 1985.

[12] Rubén de Juan-Marín, Hendrik Decker, José Enrique Armendáriz-Íñigo, José M. Bernabéu-Aubán, and Francesc D. Muñoz-Escoí. Scalability approaches for causal multicast: A survey. *Computing*, 98(9):923–947, 2016.

[13] Alan J. Demers, Daniel H. Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard E. Sturgis, Daniel C. Swinehart, and Douglas B. Terry. Epidemic algorithms for replicated database maintenance. In *6th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 1–12, Vancouver, British Columbia, Canada, August 1987.

[14] Antonio Fernández, Ernesto Jiménez, and Vicent Cholvi. On the interconnection of causal memory systems. In *19th Annual ACM Symp on Princ of Distrib Comput (PODC)*, pages 163–170, Portland, Oregon, USA, July 2000.

[15] Armando Fox and Eric A. Brewer. Harvest, yield and scalable tolerant systems. In *7th Workshop on Hot Topics in Operating Systems (HotOS)*, pages 174–178, Rio Rico, Arizona, USA, March 1999.

[16] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, 2002.

[17] James R. Goodman. Cache consistency and sequential consistency. Technical report, Number 61, IEEE Scalable Coherent Interface Working Group, March 1989.

[18] Jim Gray, Pat Helland, Patrick E. O'Neil, and Dennis Shasha. The dangers of replication and a solution. In *ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 173–182, Montreal, Quebec, Canada, 1996.

[19] Ernesto Jiménez, Antonio Fernández, and Vicent Cholvi. A parametrized algorithm that implements sequential, causal, and cache memory consistencies. *J. Syst. Software*, 81(1):120–131, 2008.

[20] Paul R. Johnson and Robert H. Thomas. The maintenance of duplicate databases. RFC 677, Network Working Group, Internet Engineering Task Force, January 1975.

[21] Rivka Ladin, Barbara Liskov, Liuba Shrira, and Sanjay Ghemawat. Providing high availability using lazy replication. *ACM T. Comput. Syst.*, 10(4):360–391, November 1992.

[22] Leslie Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE T. Comput.*, 28(9):690–691, 1979.

[23] Leslie Lamport. On interprocess communication. Part II: algorithms. *Distrib. Comput.*, 1(2):86–101, 1986.

[24] Richard J. Lipton and Jonathan S. Sandberg. PRAM: A scalable shared memory. Technical report, CS-TR-180-88, Princeton University, USA, April 1988.

[25] Prince Mahajan, Lorenzo Alvisi, and Mike Dahlin. Consistency, availability and covergence. Technical report, UTCS TR-11-22, Dept. of Computer Science, The University of Texas at Austin, USA, 2011.

[26] Francesc D. Muñoz-Escoí and José M. Bernabéu-Aubán. A survey on elasticity management in PaaS systems. *Computing*, (in press), 2016. DOI: 10.1007/s00607-016-0507-8.

[27] Douglas Stott Parker, Gerald J. Popek, Gerard Rudisin, Allen Stoughton, Bruce J. Walker, Evelyn Walton, Johanna M. Chow, David A. Edwards, Stephen Kiser, and Charles S. Kline. Detection of mutual inconsistency in distributed systems. In *Berkeley Workshop*, pages 172–184, 1981.

[28] Larry L. Peterson, Nick C. Buchholz, and Richard D. Schlichting. Preserving and using context information in interprocess communication. *ACM T. Comput. Syst.*, 7(3):217–246, 1989.

[29] Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Comput. Surv.*, 22(4):299–319, 1990.

[30] Robert C. Steinke and Gary J. Nutt. A unified theory of shared memory consistency. *J. ACM*, 51(5):800–849, 2004.

[31] Douglas B. Terry, Alan J. Demers, Karin Petersen, Mike Spreitzer, Marvin Theimer, and Brent B. Welch. Session guarantees for weakly consistent replicated data. In *3rd International Conference on Parallel and Distributed Information Systems (PDIS)*, pages 140–149, Austin, Texas, USA, September 1994.

[32] Werner Vogels. Eventually consistent. *Commun. ACM*, 52(1):40–44, 2009.

A Appendix: Formalism

This appendix provides a summary of the formalism used in [30], and justifications for those properties given in Section 3.

A.1 Specification of Consistency Properties

To begin with, some definitions are needed.

Definition A.1 (Execution)**.** *An execution is a set of processes P, a set of replicated variables V, a set of actions A, and two partial orders on A: process order ($<_{PO}$) and writes-to order ($\mapsto$).*

Definition A.2 (Memory action)**.** *A memory action is a 4-tuple (ac,i,x,v) where $ac \in A$ is r for a read and w for a write, $i \in P$ is the process submitting that action, $x \in V$ is the variable to which the operation is applied, and v is a valid value for variable x. In this context, a read means only that a write made by any other process is received and applied to the copy of x being maintained by process i. The following functions are also defined:*

action((ac,i,x,v))=ac
process((ac,i,x,v))=i
variable((ac,i,x,v))=x
value((ac,i,x,v))=v

Definition A.3 (Local order)**.** *Local order for a process i, $<_i$, is a relation such that:*
$(\forall_{a_j,a_k \in (*,i,*,*)} a_j < a_k \;\oplus\; a_k < a_j) \wedge (\forall_{x \in V, a \in (*,i,*,*)} (w, \epsilon, x, \perp) <_i a)$
where ϵ is a special symbol that does not denote any process and $\perp$ is a special value that cannot by written by any process. With this, $(w, \epsilon, x\ \perp)$ is the initial write of x.

Definition A.4 (Process order)**.** *Process order, $<_{PO}$, is the order in which actions are submitted by each process, i.e.,*

$$<_{PO} \equiv \cup_{i \in P} <_i$$

Definition A.5 (Writes-to order)**.** *Writes-to order, $\mapsto$, defines which write is read by each read, i.e.,*

$$\forall_{a \in (r,*,*,*)} \exists_{i \in P} (w, i, variable(a), value(a)) \mapsto a$$

Definition A.6 (Data order)**.** *Two actions are ordered by data order, $a_1 <_{DO} a_2$, iff $variable(a_1) = variable(a_2)$ and either*

1. *$a_1 <_{PO} a_2$, or*
2. *$a_1 \mapsto a_2$, or*
3. *$\exists_{a \in (r,*,*,*)} variable(a) = variable(a_1) \wedge value(a) \neq value(a_1) \wedge a_1 <_{PO} a \wedge a_2 \mapsto a$, or*
4. *There exists an operation a, such that $a_1 <_{DO} a <_{DO} a_2$.*

Definition A.7 (Write-read-write order)**.** *Two actions are ordered by write-read-write order, $a_1 <_{WO} a_2$, iff there exists a read r such that $a_1 \mapsto r <_{PO} a_2$.*

Definition A.8 (Serial order)**.** *A serial order, $<_{SO}$, is a minimal set of edges that enforces that $\forall_{w,r \in A}$ such that w and r are to the same variable and do not have the same value either $w <_{SO} w' \mapsto r$ or $r <_{SO} w$.*

Definition A.9 (Anti-order)**.** *Given a serial order, $<_{SO}$, $\forall_{w_1,w_2 \in A} w_1 <_{AO} w_2$ iff $\exists r_1, r_2 \in A$ such that:*

1. *$w_1 \mapsto r_1 <_{PO} r_2 <_{DO} w_2$, or*
2. *$w_1 \mapsto r_1 <_{PO} r_2 <_{SO} w_2$, or*
3. *$w_1 \mapsto r_1 <_{SO} w_2$, or*
4. *$w_1 <_{PO} r_1 <_{DO} w_2$, or*
5. *$w_1 <_{PO} r_1 <_{SO} w_2$.*

Definition A.10 (Serial view)**.** *A serial view is a total order on a subset of the actions in an execution that represents one process' view of the order in which those actions took place in memory. In a serial view, each read must read from its most recent write to the same variable.*

Taking these definitions as a base, we may specify the four consistency properties as follows:

Definition A.11 (GPO)**.** *An execution follows global process order (GPO) iff*

$$\forall_{i \in P} \exists SerialView(<_{PO} |\ (*, i, *, *) \cup (w, *, *, *))$$

Definition A.12 (GDO)**.** *An execution follows global data order (GDO) iff*

$$\forall_{i \in P} \exists SerialView(<_i \cup <_{DO} |\ (*, i, *, *) \cup (w, *, *, *))$$

Definition A.13 (GWO)**.** *An execution follows global write-read-write order (GWO) iff*

$$\forall_{i \in P} \exists SerialView(<_i \cup <_{WO} |\ (*, i, *, *) \cup (w, *, *, *))$$

Definition A.14 (GAO)**.** *An execution follows global anti-order (GAO) iff* $\exists <_{SO}$ *such that*

$$\forall_{i \in P} \exists SerialView(<_i \cup <_{SO} \cup <_{AO(<_{SO})} |\ (*, i, *, *) \cup (w, *, *, *))$$

A.2 Justification of Which Properties Are Partitionable

Let us justify whether GPO, GDO, GWO and GAO are partitionable or not.

Lemma A.1. *GPO is partitionable.*

Proof. We prove this by contradiction. Let us assume that GPO is not partitionable. This means that when a network partition occurs, two processes $p_i, p_j \in P$ that reside in different network components C_1, C_2 generate local executions whose global view does not comply with the requirements of GPO. In those executions one of those two processes, let assume p_i, has seen $a_1 = (r, i, x, v_1) <_i a_2 = (r, i, y, v_2)$ while in p_j the events that originated those reads have happened in the reverse order $b_2 = (w, j, y, v_2) <_j b_1 = (w, j, x, v_1)$.

Let us assume that the network partition happened at time t_1 and that such partition is healed at time $t_2 > t_1$. Let us analyze when those four events may have happened:

- All events happened before t_1. This is impossible, since before being partitioned this system complied with GPO and this implies that if $a_1 <_i a_2$ then $b_1 <_j b_2$, according to Definition A.11.
- All events happened after t_1 and before t_2. If p_i and p_j are in different network components this is impossible, since the network partition avoids that the writes from p_j reach p_i. Because of this, the local execution in p_i still complies with GPO on what it refers to the writes from p_j, since both processes complied with GPO before t_1. So, we reach again a contradiction.
- Events b_2 and b_1 had happened after t_1 and before t_2 and events a_1 and a_2 have happened after t_2. This is, again, impossible. Once the network partition is healed and communication between C_1 and C_2 is resumed, the consistency protocol being used for ensuring GPO propagates the updates generated in events b_2 and b_1 in FIFO order to all other network components. This means that p_i receives those updates in order $a_2 <_i a_1$. So, we reach a contradiction.
- Any other combinations may be subsumed in the previous cases or clearly break Definition A.10.

This proves the lemma. □

Lemma A.2. *GDO is not partitionable.*

Proof. The proof given in Theorem 1 also proves this lemma. □

Lemma A.3. *GWO is partitionable.*

Proof. We prove this by contradiction. Let us assume that GWO is not partitionable. This means that when a network partition arises two processes $p_i, p_j \in P$ that reside in different network components C_1, C_2 generate local executions whose global view does not comply with the requirements of GWO. In those local executions one of those two processes, let assume p_i, has seen $a_1 = (r, i, x, v_1) <_i a_2 = (r, i, y, v_2)$ while in p_j the events that originated those reads have happened in the reverse order $b_2 = (w, k, y, v_2) \mapsto b_r = (r, j, y, v_2) <_j b_1 = (w, j, x, v_1)$.

Let us assume that the network partition happened at time t_1 and that such partition is healed at time $t_2 > t_1$. Let us analyze when those four events (a_1, a_2, b_2, b_1) may have happened:

- All events happened before t_1. Before being partitioned this system complied with GWO. This implies that if $a_1 <_i a_2$ then $b_1 \mapsto b_{r'} <_k b_2$, according to Definitions A.13 and A.7, contradicting the assumption.

- All events happened after t_1 and before t_2. If p_i and p_j are in different network components this is impossible, since the network partition avoids that the writes from p_j reach p_i. Because of this, the local execution in p_i is a GWO-compliant prefix on what it refers to the writes from p_j, since both processes complied with GWO before t_1. Process p_i has not seen any of those writes in a sequence that violates the $<_{WO}$ order (nor in any other, actually). So, we reach again a contradiction.

- Events b_2 and b_1 had happened after t_1 and before t_2 and events a_1 and a_2 have happened after t_2. This is, again, impossible. Once the network partition is healed and communication between C_1 and C_2 is resumed, the consistency protocol being used for ensuring GWO propagates the updates generated in events b_2 and b_1 in causal order to all other network components. This means that p_i receives those updates in order $a_2 <_i a_1$. So, we reach a contradiction.

- Any other combinations may be subsumed in the previous cases or clearly break Definition A.10.

This proves the lemma. □

Lemma A.4. *GAO is not partitionable.*

Proof. This is derived from the fact that GAO is a stronger property than GDO. This means that all GAO executions comply with the GDO requirements. Since no GDO execution is partitionable, then no GAO execution will be. □