Research statement and overview

Marc Shapiro
UPMC-LIP6 & INRIA
http://lip.fr/Marc.Shapiro/

6 February 2017

1 Research overview

Large-scale distributed computing systems are central to our modern economies, witness the Internet, cloud computing, peer-to-peer computing, parallel computing, nomadic and pervasive computing, etc. Advances in this area impact the economics and user experience of large segments of society.

My research focus is on *sharing mutable information* in large-scale distributed systems, i.e., ones with high latency, subject to partial failures, and comprising large amounts of data. This is relevant to, for instance, enterprise intranets, social or co-operative networks, distributed databases, and mobile computing.

This includes an interest in concurrent algorithms (especially fine-grain algorithms for multicore computing), and in memory management and garbage collection algorithms (virtual memory, distributed GC, GC in the presence of weak consistency, and multicore GC). This is relevant to modern operating systems and Managed Runtime Environments (MREs, a.k.a. language virtual machines).

1.1 Approach and impact

My approach combines the practical and the theoretical.[1] Result-oriented research, typical of the systems community, was published in venues such as ASPLOS [23, 24], EuroSys [7], OSDI [17], SOSP [1], SRDS [8, 45, 50, 72], Middleware [4, 82], ICDCS [3, 19, 43, 48, 49, 51], Systor [74], LADIS [5, 6, 34, 67], or Computing

[1] Herein, "I", "we," "my" or "our" are to be read as short for "my co-authors and myself." Citation counts, marked GS, were compiled from Google Scholar on 20 December 2016. My publications are accessible at http://lip6.fr/Marc.Shapiro/pubs.html.

Systems [57]. I take a *principled* approach, isolating and formalising the fundamental problems and formulating practically-relevant theory, also publishing in more theoretical venues of our domain, such as POPL [25], PODC [30, 31, 58], DISC [11, 54, 61], OPODIS [2, 62], or Euro-Par [46, 70]. My work impacts other research communities, as evidenced by publications in the areas of: parallel computing and concurrency [80: PPoPP]; collaborative work [14, 28: Group and CollaborateCom]; programming languages [25: POPL], [33: PLDI], [68: Concur], [20, 56: ECOOP]; artificial intelligence and machine learning [27: IJAIT]; and databases [9, 69: Data Engineering Bulletin, BDA].

I gave invited presentations to venues such as: Concur 2016, U. Joseph Fourier 2016, Inria-EPFL Workship 2016, CodeMesh 2015, Royal Holloway London U. 2015, W. on Chemistry of Concurrent and Distributed Programming 2015 and 2011, CurryOn 2015, Technion Systems Day and Technion Colloquium, both in June 2014 (via the ACM Distinguished Speakers Program), Darmstadt Technical University, MAKI Distinguished Lecture Series, in June 2013; $(EC)^2$, the Int. W. on Exploiting Concurrency Efficiently and Correctly, co-located with CAV 2013; the Verico 2011 workshop (Verification of Concurrent Data-Structures) co-located with POPL 2011; WetICE 2011, the Int. Conf. on Collab. Tech. and Infrastructures; the Mysore 2011 workshop on *The Chemistry of Concurrent and Distributed Programming*; ICFP 2006, the major conference on functional programming [53]. Furthermore, I initiated the Dagstuhl Seminar on Consistency in Distributed Systems (Feb. 2013), in order to bring together the researchers from diverse communities, i.e., distributed systems, distributed algorithms, cloud computing, parallel computing, distributed databases, and parallel architectures [29]; a new edition has been accepted for Feb. 2018. I recently initiated the PaPEC/PaPoC series of workshops (Principles and Practice of Data Consistency), to which I was a frequent contributor [15, 40, 78, 79, 83].

The *proxy* concept that I invented [49: ICDCS 1986] is ubiquitous, e.g., all over in the World-Wide Web.

The CRDT concept, a data model that I invented in order to simplify the correct design and implementation of available applications [65, 66] is creating a lot of excitement, including in industry; for instance, the Riak distributed datastore has support for CRDTs.

My ten most-cited papers [1, 30, 41, 43, 47, 49, 57, 58, 64, 65] total 2644 Google Scholar (GS) citations. GS reports that my papers were cited a total of 4902 times, and gives an H-index of 29. The 38 papers available from the ACM Digital Library were downloaded 16 923 times.

I am the only French author to have published at both SOSP [1] and OSDI

[17].[2]

The Inria research in the area of Distributed Systems and Services was evaluated in October 2016 by an external evaluation panel. Their assessment of the Regal research group contains the following:

> *Regal has generated a number of notable papers that have gained traction in the academic community, for example the work on failure detectors and on CRDTs. The panel emphasizes that the work on CRDTs typifies what we expect from great research institutes: original ideas that defy the status quo and move the field towards new directions, opening the door for new classes of research results. Multiple panel members expressed the view that when first hearing about CRDTs, they dismissed the idea as "crazy." Yet, the work done by the REGAL team has not only convinced skeptical academics that this is actually a good idea, but also managed to get traction among developers of real systems. INRIA should be very proud of what they have achieved.*
>
> *[...] The partnership with Basho Technologies on the implementation of CRDTs in the cloud database Raik* [sic] *is a an enormous outcome, with many users of Raik now using them. Finally, the adoption of CRDTs in the infrastructure of large enterprises such as Facebook and TomTom highlights the true depth of impact and partnership that the team has with industry. This impressive performance in industrial collaboration is further strengthened by participation in a large number of EU projects.*

The panel members were Dilma Da Silva (Chair), Salima Benbernou, Yolande Berbers, Ewa Deelman, Maurice Herlihy, Flavio Junqueira, Antony Rowstron, and Liuba Shrira.

1.2 Research Perspective

Fueled by the proliferation of mobile devices and of cloud services, computing is undergoing major changes. On the one hand, data is becoming more decentralised, across numerous private, mobile, capable devices, that often hold the most recent, authoritative updates. On the other, data is increasingly produced, processed, and stored in the cloud.

As users are more and more interconnected in large-scale social networks, the issues of concurrency, latency tolerance, consistency and security become more

[2] The premier publication venues for the systems community are SOSP, OSDI and EuroSys.

acute. The amount of available user data increases steeply, and privacy issues cannot be ignored.

Large-scale services need to decentralise, in order to tolerate latency and to ensure availability in the presence of network partitions. Therefore, architects of cloud services such as Amazon, Google or Facebook have turned to optimistic replication and eventual consistency, validating our long-term vision. However, they have been using ad-hoc, unprincipled approaches, that provide little in the way of useful guarantees. Application programmers remain unable to comprehend the costs and consequences of a design decision.

Recently, the pendulum has been swinging back to strong consistency, because weak forms of consistency appear anomalous to application developers. However, weaker consistency can result in order-of-magnitude cost/energy/performance improvement [4]. There is no one-size-fits-all solution, and the strong/weak consistency trade-offs are here to stay.

Our thesis is that this bottom-up approach, shoe-horning an application to a predefined consistency model, is the wrong way to look at the problem. Instead, we would like to start from the *requirements of the application* and to *tailor consistency precisely to application requirements*.

Our approach starts with a data model for replicated systems that provides both backward compatibility with sequential systems and correctness. Our Conflict-Free Replicated Data Types (CRDTs) have a standard sequential semantics and provably converge under concurrent updates. Applications invoke transactions, which perform multiple reads and updates in a consistent and all-or-nothing fashion.

Next, we examine what application-level properties of the data (or *invariants*) the database can guarantee while remaining available. These include *Equivalence* and *Partial-Order* relations between objects. Operations that impact only Equivalence and Partial-Order invariants can be commute (and hence be available), as long as the system supports transactions and causal consistency respectively. Therefore, we advocate ensuring Transactional Causal Consistency (TCC) as the baseline model. TCC is the strongest model that does compromise availability, and ensures both Equivalence and Partial-Order type invariants.

TCC is not sufficient to ensure sequential correctness in all cases, which is why the CAP result is relevant; however, it is often overkill to impose a total order of all operations. To address this issue, we use the CISE static analysis tool, which checks for the remaining *Precondition-Stable* type invariants in the context of a concurrency control model. If the check is positive, this proves that the consistency model is strong enough to guarantee the application invariants. If not,

the tool provides guidance that helps the application developer, either weaken the semantics, or strengthen the concurrency control, to precisely the amount required by the application specification.

By combining TCC with CISE, we prove that all the application's sequential invariants are safe. We ensure the best possible availability (and hence performance) that is allowed by the application semantics. At the same time we ensure that synchronisation is strong enough to guarantee safety, without requiring any guesswork by the application developer. The system is as available as possible, and as consistent as necessary.

There remain many challenges in supporting this hybrid model and in scaling the database to hundreds of thousands of database replicas at the edge, in order to support the requirements of so-called "Fog Computing." One is how to implement Transactional Causal Consistency at such a scale, without excessive size of metadata and without excessive delays; and, in particular, how to ensure partial replication without impacting correctness. This is likely to require some kind of hierarchical partitioning of the database, which must be adaptive as usage patterns change over time. Another challenge is ensuring security guarantees (which can be viewed as a specific kind of invariant) despite weak consistency. We also believe that scalability and the need to tailor consistency to application requirements will be helped by decomposing consistency models into (mostly-)orthogonal primitives, which can be implemented as (mostly-)independent modules.

An important operational requirement is to improve the visibility of, and control over, the actual workings and behaviours of distributed systems. Beyond our existing CISE analysis, we need better tools for proving correctness and removing errors. We need tools for deploying and monitoring distributed systems; tools for generating and executing system and application tests; and tools that help analyse the performance and diagnose bottlenecks.

The area of replication and consistency has received much research attention. However, it is still not well understood by practioners. We note that, even in research circles, the guarantees and subtleties of weaker models are not receiving much attention. Yet, a better understanding of consistency is essential to building and using massive-scale systems at an acceptable cost. My current research agenda is to improve this understanding, first by decomposing consistency models along orthogonal axes, and second by identifying the invariants that are guaranteed along each point of the axes. Eventually I hope to be able to use this knowledge constructively, creating consistency protocols tailored to specific uses by composing primitives.

2 Current research: bridging the CAP gap between correctness and availability

Application design includes deciding on a data schema and on a set of transactions, and identifying *data invariants*, i.e., properties that the stored data must maintain. Even in a sequential execution, updates must be written carefully to maintain the invariants. For instance, for a bank to maintain the invariant that accounts are never non-negative, the *withdraw(amt)* operation must check that *amt* is less than the current balance.

Thus, we make the crucial assumption that, if in some state of the database the data invariants are true, every individual update operation will leave the invariants true. We say that application operations are *correct in isolation* (the "C" in ACID).

The challenge is to maintain the same invariants when running above a database with concurrency. We can now define precisely what we mean by the *guarantee* of a consistency model. If the application is Correct-In-Isolation, the invariants are true in any sequential execution; sequential execution guarantees all invariants. The guarantee of a consistency model is the class of invariants that remains true at all times, transparently and without requiring any extra involvement by the application programmer.

Only the strongest consistency model, Strict Serialisability (SSER), guarantees all classes of invariants. Although distributed implementations of SSER exist (e.g., Google's Spanner), their availability, performance, scalability, and monetary cost are questionable. These constraints imply to use weaker models, yet any weaker model has *anomalies* that break some invariants.

Indeed, the CAP Theorem points to an inherent conflict between consistency guarantees on the one side, and availability and performance on the other. We propose to bridge this CAP gap, by understanding which application invariants are compatible with availability, and tailoring the consistency model precisely to the application requirements. The good news is that correctness and high availability do not necessarily conflict. The bad news is that this cannot be transparent, and requires a deeper understanding of the application's invariants. We propose a new methodology, *Just-Right Consistency*, to match consistency and application requirements, synchronising only when strictly necessary, based on a novel static analysis (Section 2.3).

2.1 Conflict-Free Replicated Data Types (CRDTs)

The standard data model, based on registers (read/write memory cells) does not support concurrency well. The first order of business is to provide a suitable data model that supports concurrent updates, while minimising exposure to concurrency anomalies.

For instance, we can represent updates to a shared counter, not as writes but as addition and subtraction. Since these operations commute, all replicas that receive the same updates converge, even if received in different orders. We generalise this insight to a data model designed for asynchronous replication, called *Conflict-free Replicated Data Types*, or CRDTs.

A CRDT is a pre-packaged replicated data type that encapsulates replication and concurrency. Replicas of a CRDT can be updated in parallel without synchronisation, and are guaranteed to converge to a sensible value, without loss. CRDTs come in different flavours. In addition to counters, we have designed CRDT sets, maps, graphs, sequences, and specific kinds of registers.

A well-designed CRDT behaves just like its counterpart sequential datatype, when used sequentially. If two updates commute in the sequential specification, then the same updates concurrently converges to the same result as the sequential execution. The key challenge in CRDT design is providing a sensible semantics for concurrent updates that *do not commute*. For instance, for an object of type Set, there is no obvious "right" semantics for insert (e) || remove (e) (concurrently adding and removing the same element). In fact, any outcome that is deterministic and does not depend on the order of delivery (or any other external factor) will provide the convergence property. Therefore, our library provides different CRDT set variants that differ only in the result of insert (e) || remove (e). In the *Add-Wins* semantics, insert takes precedence over a concurrent remove of the same element. Using the *Last-Writer-Wins* type, the operation that happened at the highest clock time takes precedence. It is up to the application developer to decide which of these variants is most appropriate for her application.

The intuition behind CRDTs previously appeared in the research folklore, but my co-author Nuno Preguiça and I were the first to conceptualise the idea and to explore its mathematical properties. The state space of a CRDT data type is a semi-lattice; updates move state forwards in the semi-lattice, and merging replicas is the least-upper-bound operation. In an alternative (but equivalent) view, a CRDT is an object whose concurrent updates commute. Our first CRDT design was Treedoc, a simple and available solution to the problem of decentralised concurrent editing [43: ICDCS 2009, GS=130]. We formulated the principles of CRDTs at SSS 2011 [65: GS=220]. We developed a collection of useful CRDT

designs [66: BEACTS, GS=82], [64: Inria RR, GS=152]. Our primary example is the OR-Set, an add-wins set that we show to be correct [11: DISC 2012]. Recently we revisited CRDT registers [83] and efficient sequence designs for concurrent editing [14].

Application developers build their applications by using CRDTs in a similar way they use traditional abstract data types. In addition, selecting a specific CRDT determines how their applications behave when concurrent updates occur.

CRDTs have been adopted by industry, for instance, Basho's `riak` database or TomTom's NavCloud, and prompted publications by such eminent researchers in the Distributed Systems and Programming Language and Verification communities as Rodrigo Rodrigues, Peter Bailis, Hagit Attiya, Eric Brewer, Marcos Aguilera, Peter Van Roy, Sebastian Burckhardt, Alexey Gotsman or Ahmed Bouajjani.

Taken together, my papers on CRDTs total 668 GS citations [11, 12, 43, 55, 64–66, 81].

2.2 Available invariants

We identify two classes of invariants that a database can guarantee without compromising availability: type EQ and PO.

2.2.1 EQ: Equivalence-type invariants between data items

One common case is to maintain some kind of equivalence relation between data items. For example, in a social network, ensuring that if A is a friend of B, then B is also a friend of A. Modifying one of the items requires to modify the other "at the same time." The application groups these updates in a *transaction*. The database ensures that a client observes either all of the transaction's updates (even if made to different servers), or none. This *All-Or-Nothing* property (the "A" property in ACID) does not require synchronisation. If two replicas cannot communicate, then one does not see the updates at the other; if they can, the whole all-or-nothing package is visible. Either way, both sites remain available to their local users.

The converse property to All-or-nothing, often overlooked, is to ensure that reads are consistent within a transaction. To ensure this, transactions have the *snapshot* property, whereby all the reads of a transaction. Like all-or-nothing, the snapshot property does not compromise availability.

2.2.2 PO: Invariants based on a partial order

A second class of highly-available invariants is based on a partial order. Examples include *numerical inequality* of the form $A \geq B$. Another example is an implication; for instance, a social network where user A can post on B's wall only if they are friends: $posted(T, A, B) \Rightarrow friend(A, B)$. A very common case is *referential integrity*, of the form $f(x) \in A \Rightarrow x \in B$ where A and B are tables.

The general approach to maintaining this kind of invariant is to perform updates "in the right order," for instance by following the Demarcation Protocol. If some replica could observe the updates in a different order, the invariant might be violated. Thus, to guarantee this class of invariants, the database should ensure *causal consistency*: if some event happens before another, then all replicas must observe them in that same order. In contrast, unrelated or *concurrent* events can be observed in any order.

Many models, including for instance Serialisability, ensure a slightly weaker guarantee, *transitive visibility*: if some transaction T2 reads the result of a transaction T1, then all replicas observe those events in the order T1 followed by T2. However, if T2 does not read from T1, the transactions are considered unrelated, *even if the same client performed them in some order.*

The order is typically implemented by piggy-backing ordering information on top of update messages, and not making visible a received update unless its dependencies have themselves been made visible. This does have some overhead, but does it not impact availability.

2.3 Non-available invariants: PS, the CISE analysis, and Just-Right Consistency

We now address invariants that are *not* compatible with availability. They are what makes the CAP theorem relevant.

We assumed earlier that application updates are correct-in-isolation. Each update operation must test some *preconditions* to ensure that it will leave the invariants true. For instance, before accepting a withdraw(amt), the bank application checks that amt *le* balance, in order to ensure the invariant balance ≥ 0. Let us call this class of invariants *Precondition-Stable* (PS).

This check can be fooled by replication and concurrency. The problem is that this precondition is not *stable*, i.e., between the moment where the application checks the precondition to true, and the moment where it subtracts the amount, the precondition may have become false because of a concurrent withdrawal.

For instance, in the bank application, concurrent withdrawals are problematic, but concurrent deposits can never violate the invariant. Furthermore, not all withdrawals will lead to a violation; for example, if the initial balance was €10,000, concurrent withdrawals of €1 will not be a problem.

A first approach is to move the necessary synchronisation out of the critical path by batching and using escrow techniques. The bank example generalises to numeric invariants used by many applications; therefore we have devised a new Bounded Counter CRDT [8: SRDS 2015, GS=10].

More generally, we wish to synchronise operations only if this is required to maintain the invariant. However, this is tricky: too little synchronisation and the database gets corrupted; too much and the application grinds to a halt. Getting this right requires a detailed knowledge of the application.

To ensure PS invariants, our insight is that it is not necessary to order all operations: *ordering is necessary only to prevent unstable concurrency* or non-commutative operation. We presented this idea informally at EuroSys 2015 [7: GS=26], and formalised and proved its soundness at POPL 2016 [25: GS=17]. We have implemented this so-called CISE logic in a tool [39, 40], thus enabling the application developer to ensure that her application is both correct, and as available as possible given the invariant.

2.4 Stronger guarantees for available consistency: Transactional Causal Consistency

We have identified two classes of invariants that can be guaranteed transparently in an available system: EQ invariants using transactions, and PO using a transitive- or causal-visibility model.

Transactional Causal Consistency (TCC) ensures both. *Without* TCC, some very basic expectations about the system are violated. It's very hard for a developer to understand what's happening if updates are delivered in anomalous order, and many applications have PO invariants such as referential integrity. Similarly, without transactions, it is very difficult to maintain relations between different objects correctly.

TCC is the strongest model that does not compromise high availability.[3]

Causal consistency maintains a (partial) order of events. This requires to piggy-back ordering meta-data on top of messages. In many cases, the size of

[3] Previous authors have called causal visibility the strongest available model, but they overlooked EQ invariants and transactions.

meta-data can become intolerable, for instance a vector clock with thousands of entries, one per process. We have devised a number of techniques to keep this cost down, by leveraging a constrained communication topology [82: Middleware 2015, GS=17] and/or strongly-consistent clusters [3: ICDCS 2016, GS=4].

The all-or-nothing property comes almost for free in causal consistency, simply by giving the same timestamp to all the updates of a transaction.

However, the snapshot property raises difficulties in a partitioned/sharded database. Naïve approaches tend to synchronise, which should be avoided; we have identified a trade-off between availability and staleness, which we leverage and try to close [3, 78, 82].

Our PhysiCS protocol provides efficient and scalable support for consistent snapshots for a number of transactional protocols, for instance NMSI (non-monotonic snapshot isolation), a causally-consistent variant of Snapshot Isolation [79]. It is the first protocol to rely on a single scalar taken from a physical clock for tracking causal dependencies and building causally consistent snapshots. Its commit protocol ensures atomicity and the absence of write-write conflicts. THE PhysiCS approach increases concurrency and reduces abort rate and metadata overhead as compared to state-of-art systems.

For geo-replication to scale beyond a few data centres, and in particular to be able to cache data at the edge, requires *partial replication*. This means that a DC will replicate only the portion of the data that is of interest to it; for example in a key-value store, the DC might store values corresponding to a portion of the keys only. Under partial replication, it is particularly challenging to ensure causal consistency guarantees in an available way, since a partial replica may lack some of the information required to ensure the transitivity of causal order, and fetching that data on demand impacts availability. We are working on designing a causally consistent protocol for geo-distributed partial replication [15]. Our key insight is to move the burden of dependency checking, away from a DC receiving an update, which may lack information, to the originating DC. An update occurring at some server of one DC is sent directly to its sibling replica(s) at other DCs, but it is not made visible to readers at the receiving DC until the origin DC confirms that its dependencies have been received.

2.5 AntidoteDB

The Antidote database [75] is a geo-replicated database for geo-cloud systems, designed to bridge the CAP gap. It stores data on behalf of applications in a highly parallel and highly distributed fashion. It has an available architecture, to be as efficient, scalable and low-cost as possible. Antidote scales from single-machine

to geo-replicated deployments. Similarly to a Content Delivery Network (CDN) it improves application responsiveness, performance and scalability by replicating data near the user; whereas a CDN does this for restricted kinds of content, Antidote performs a similar feat for mutable database data.

Antidote's data model builds on an extensive library of CRDTs, including not only the familiar Last-Writer-Wins and Multi-Value registers, but also counters, sets, maps, sequences, and bounded counters. It currently supports clients written in JavaScript, Java and Erlang; any other language can be supported through its protocol-buffer interface. Unlike many recent cloud databases, there are no restrictions on transactions, i.e., a transaction will consistently read or update any number of CRDT objects, not necessarily known in advance (so-called "interactive" transactions) [3: ICDCS 2016].

The base consistency model is TCC, which provides the strongest guarantees for an available system; optionally, an individual transaction can run seamlessly in synchronous mode, in order to provide as much consistency as necessary. Within a DC, independent shards are assigned disjoint sets of keys; a transaction involves only those shards that are accessed by the transaction, thus allowing disjoint transactions to run in parallel. A TCC transaction never aborts (only the optional synchronous transactions can abort).

Antidote maintains multiple versions of an object; recent versions are cached in memory and can be accessed concurrently without blocking. Antidote's TCC semantics and its internal design tolerate delayed or duplicate messages, as well as long periods of disconnection, without impacting the performance of local transactions. The meta-data required to maintain causal consistency is "lean but safe" [82]; it consists of vector clocks with one entry per DC; therefore, we expect Antidote to be able to scale to several dozens of DCs.

Antidote is currently undergoing an extensive experimental performance measurement campaign, using the FMKe benchmark application (modeled after the Danish Healthcare network FMK). Initial results confirm that Antidote has transaction latency of the order of milliseconds (depending on the number of reads and writes in the transaction). Throughput scales well, from a maximum of approximately 10 000 FMKe transactions/s with a single DC of four Antidote servers, to 19 000 FMKe transactions/s with three DCs of four servers.

Antidote, CISE, and FMKe were developed in the EU-funded project SyncFree, which I led; see Section 4.3 for more information.

2.6 Consistency in 3D

Comparisons of different consistency models often try to place them in a linear strong-to-weak order. However this view is clearly inadequate, since it is well known, for instance, that Snapshot Isolation and Serialisability are incomparable. In the interest of a better understanding, we propose a new classification, along three dimensions, related to: a total order of writes, a causal order of reads, and transactional composition of multiple operations. A model may be stronger than another on one dimension and weaker on another. We believe that this new classification scheme is both scientifically sound and has good explicative value. We study the three-dimensional design space intuitively, placing a number of well-known models in this 3D space, and relating the degrees of each axis with the invariant classes that they guarantee.

This work was presented at Int. Conf. on Concurrency Theory (CONCUR 2016) as an invited keynote talk [68].

2.7 Merging Semantics for Conflict Updates in Geo-Distributed File Systems

In cooperation with the Scality SA company, a large-scale software-defined storage provider, we have studied how to design a geo-distributed file system. The system should be highly available and should conform closely to the POSIX semantics. We leveraged CRDTs and the CISE analysis. Our file system model, and our merging semantics for resolving conflict updates, fully describes a Posix-like file system, with all of its components including hard links. This model is able to identify all conflict cases which are classified into direct, such as concurrent updates to the same file, and indirect, such as cycles in the namespace of the file system. The merging semantics resolve all types of conflicts while being able to preserve the effect of all conflict updates. Our implementation of the system and the merging semantics outperforms the existing systems in terms of feature completeness.

This work was presented at Systor 2015 [74: GS=8].

3 Other contributions

3.1 1976–1990: Proxies and fragmented objects

My initial work focused on patterns and structures for programming distributed systems, which were at the time a novel idea. I proposed the concept of a *proxy* [49:

ICDCS 1986, GS=390]. Proxies implement the *Fragmented Object (FO)* concept. An FO is a single distributed object with local representatives on each node. This structure, on the one hand provides transparency for clients (they only talk to the local proxy), but on the other also provides the system developer with flexibility to design and place distributed components (since the client does not observe them) [36]. My group designed the SOS operating system [57: Computing Systems 1989, GS=154] supporting FOs (and implemented using FOs).

SOS was influential and generated a number of publications. It provided the basis for subsequent research on communication protocols [35], on programming languages for distributed computing [56: ECOOP 1989] [26], and on object persistence and migration.

SOS was developed within the European Commission project SOMIW. Proxies are now ubiquitous in all distributed systems, e.g., on the WWW. SOS influenced several later systems, e.g., the COOL (Chorus Object-Oriented Layer) product of Chorus Systèmes, Guide, Globe or Amadeus.

3.2 1989: Chorus virtual memory subsystem

I participated in the design and formalisation of the Chorus virtual memory subsystem. This system was remarkable in offering a unified interface and implementation for diverse hardware and software requirements. The paper was published at SOSP 1989 [1: GS=147].

3.3 1990–1999: Distributed references and garbage collection

Remote, mobile and persistent objects, as pioneered in SOS, proved to be a generally-useful idea. To support them, I designed a distributed referencing mechanism called Stub-Scion Pair Chains (SSPC) [58, 59: PODC+TR 1992, GS=226]. SSPCs are capable of referencing remote and mobile objects and support automatic distributed garbage collection. They were designed to be efficient and robust. Rebinding was added later, to deal with dynamic situations [51: ICDCS 1994].

Garbage collection (GC) automates object management and persistence, but it is difficult in distributed systems [41]. Building on SSPCs, I published extensively on GC in the classical distributed-system model, i.e., disjoint nodes communicating by asynchronous messages [50: SRDS 1991], [58: PODC 1992], in particular the difficult challenge of collecting distributed cycles of garbage [33: PLDI 1998, GS=31] [32, 60].

This topic is an excellent example of the interplay between fundamental research and practical impact. SSPCs were designed to fulfill a practical need, but the mechanism and the garbage collection algorithms were designed from first principles.

Dolev's PODC 2006 keynote cites Reference 58 as a "Century Paper." SSPCs influenced systems such as Globe, implementations of Corba and Java RMI, and is widely cited in papers on distributed objects and distributed garbage collection. SSPCs were developed under EU Esprit funding (project Harness), under a grant from Novell Unix Systems Laboratory, and under a contract with CNET.

3.4 1993–2000: Garbage collection in a cached memory

Distributed systems are often based on a shared-memory model, using pointers and not remote references. In practice, memory replication and caching algorithms cannot guarantee absolute consistency.

This departs radically from the classical message-passing model of distributed systems, and requires a completely new approach to garbage collection. The Larchant system, which I designed with Paulo Ferreira, [16, 18] was the first to do so. The following pragmatic decisions ensured efficiency and scalability [17: OSDI 1994, GS=61]. Larchant GC combines tracing and counting. Memory is divided into units of replication. Each unit maintains meta-data enabling the GC to trace its pointer graph independently of other units. When a unit is modified in memory, the next run of tracing GC modifies the associated meta-data, setting off counting for downstream units. Cross-unit cycles of garbage are collected by tracing multiple units opportunistically as they are present in a same cache.

With this design in mind, I formulate a set of general safety invariants [19, 20, 54, 60]. To ensure that GC is independent of consistency, the "Union Rule" states that an object may be collected only if it is not reachable in the union of local pointer graphs. GC events and consistency events are locally ordered, to ensure that pointer mutations are not missed, while still allowing the GC to be mostly decoupled from consistency. Message ordering rules ensure that objects are not collected prematurely.

Based on the Larchant results, from 1997 to 2000, I was the Principal Investigator of the European Long-Term Research project PerDiS "A Persistent Distributed Store for Collaborative Engineering Applications." This system supports direct and efficient sharing of data between applications, even those executing at different times or different locations [13, 21]. Memory accesses are transactional, memory is replicated consistently and is garbage collected. PerDiS supports secure

data sharing and is fault tolerant. The persistent memory was used for a suite of cooperative CAD applications for the building industry.

A paper on Larchant was published at OSDI 1994 [17]. Larchant received a grant from Digital Equipement Corporation. PerDiS was an EU Long-Term Research project, of which I was the leader.

3.5 1998–2005: Disconnected operation and reconciliation

Users co-operating over PerDiS requested the ability to work in isolation. This requires disconnected operation and reconciliation of update conflicts. I was thus motivated by practical needs to study the fundamentals of optimistic replication [47: Computing Surveys 2005, GS=608]. I identified conflicts as the symptom of a violation of application invariants. To avoid them, the proposed solution is to reify application invariants, i.e., make them first-class objects, called constraints [30: PODC 2001, GS=200]. This enables the scheduler to avoid schedules that would lead to conflicts, in a principled, flexible, general manner. This concept was implemented in the IceCube system [42: CoopIS 2003, GS=70], a general-purpose, multi-application reconciler, parameterised by application semantics [63], for disconnected operation.

IceCube was developed at Microsoft Research Cambridge. I was invited to give a keynote presentation on this work at BDA 2002 [52].

3.6 2004–2013: More available strong consistency

Strong consistency aims to maintain strong system invariants by disallowing concurrent conflicting updates, or equivalently ensuring a total order of updates. I have taken a principled approach to improving its performance, scalability and availability while ensuring strong consistency, by leveraging application semantics. (See also next section.)

My Action-Constraint Framework (ACF) provides a framework for expressing specific requirements and behaviours in replication and consistency, by reifying operations as actions and invariants as constraints. It allowed us to study pessimistic vs. optimistic algorithms, and total vs. partial replication [62: OPODIS 2004, GS=35].

The Telex system is an application platform system, parameterised by ACF annotations that formalise the application semantics [10]. Telex adapts consistency to application requirements, but contrary to our later work (based on static

analysis, described in Section 2.3) it requires detailed developer input and relies in part on run-time heuristics.

With my student Pierre Sutra, I designed novel consensus algorithms that leverage the results of speculative execution [73], minimise aborts and roll-backs [69], remain safe in the presence of partial replication [70], or leverage operation semantics [71]. A number of applications have been developed above Telex, including a cooperative model editor [38].

We developed a consensus algorithm leveraging commutativity, called Fast Genuine Generalized Consensus (FGGC) [72: SRDS 2011, GS=9]. FGGC the first such protocol to reach the theoretical minimum, and as a consequence, it outperforms Generalized Paxos by a wide margin on a WAN.

Two promising approaches to ensure scalability under strong consistency are Snapshot Isolation (SI) and Genuine Partial Replication (GPR). SI removes synchronisation on reads, and it aborts only write-write conflicts; GPR ensures that independent transactions do not need to communicate.

With Pierre Sutra and Masoud Saeida Ardekani, we established an important impossibility result: with interactive transactions (i.e., whose read- and write-sets are not known in advance), SI and GPR are mutually exclusive [46: Euro-Par 2013]. We (Masoud, Marek Zawirski and myself) also showed how different snapshot models impact the performance trade-offs [44]. Accordingly, we designed a novel SI-like consistency model, Non-Monotonic Snapshot Isolation (NMSI), that is compatible with GPR and that leverages a consistent but liberal snapshot model; it performs orders of magnitude better than previous models such as SI or PSI (Parallel Snapshot Isolation) [45: SRDS 2013]. In fact its performance approaches that of weak consistency.

We also built a modular framework for composing (with only a few hundred lines of code) and comparing protocols fairly [4: Middleware 2014, GS=8]. We measured orders-of-magnitude scalability difference between strong and weak protocols and identified the sources of these differences.

3.7 2004–2015: Fine-grain concurrency and garbage collection in large-scale multicore computers

In a related but separate research thread, I work on algorithms for advanced multicore computers, fine-grain concurrency, garbage collection, and persistence.

While working at MSRC, I collaborated with Maurice Herlihy, Tony Hoare and Victor Vafeiadis on design and proof methods for concurrency. It is very difficult to reason about interference between threads in fine-grain shared-memory

concurrent programs, because of combinatorial explosion and non-determinism. However, well-constructed programs use some form of concurrency control to avoid dangerous interleavings. Taking this structure into account simplifies reasoning. To verify this insight, we studied a family of fine-grain implementations of a linked list [80: PPoPP 2006, GS=77]. The "Rely-Guarantee" approach is well adapted to both informal and formal reasoning [53].

I was invited to give a keynote speech on this subject at ICFP 2006 [53].

With my colleagues Gaël Thomas and Julien Sopena, and our student Lokesh Gidra, I recently addressed the issues of scalable concurrent garbage collection algorithms for NUMA computers.

On contemporary cache-coherent Non-Uniform Memory Access (ccNUMA) architectures, applications with a large memory footprint suffer from the cost of the garbage collector (GC), because, as the GC scans the reference graph, it makes many remote memory accesses, saturating the interconnect between memory nodes. We address this problem with NumaGiC, a GC with a mostly-distributed design. In order to maximise memory access locality during collection, a GC thread avoids accessing a different memory node, instead notifying a remote GC thread with a message; nonetheless, NumaGiC avoids the drawbacks of a pure distributed design, which tends to decrease parallelism. We compare NumaGiC with Parallel Scavenge and NAPS on two different ccNUMA architectures running on the Hotspot Java Virtual Machine of OpenJDK 7. On Spark and Neo4j, two industry-strength analytics applications, with heap sizes ranging from 160 GB to 350 GB, and on SPECjbb2013 and SPECjbb2005, NumaGiC improves overall performance by up to 45% over NAPS (up to 94% over Parallel Scavenge), and increases the performance of the collector itself by up to 3.6× over NAPS (up to 5.4× over Parallel Scavenge).

This work was published at PLOS 2011 [22: GS=25], ASPLOS 2013 [23: GS=26] and ASPLOS 2015 [24: GS=12].

4 Professional activities and service

4.1 Community service

I am a member of the ACM since 1981. I am a Senior Member of the ACM and a speaker for ACM's Distinguished Speaker Program (DSP). I was previously a member of ACM's Distinguished Service Award committee.

For many years I have been at the forefront of growing and organizing the CS/Informatics community, which is still extremely fragmented in Europe.

As Vice-Chair of SIGOPS, I was instrumental in bringing SOSP to Europe for the first time in 1997, and again in 2005.

I created and chaired the French chapter of ACM Sigops in 1996.

In 2004, I created EuroSys scientific society, the pan-European chapter of ACM Sigops, which I chaired for five years. In 2006 I initiated the EuroSys conference, which has grown to be one of the top systems conferences.

I was instrumental in creating the ACM Europe Council, of which I was a member since its inception in 2008, until 2015. I initiated and ran the 2011 Workshop of European ACM Chapters, and recently created the ACM Council of European Chapter Leaders.

I was recently elected a member of the board of Société Informatique de France (SIF), the French learned society in Informatics. I am in charge of research, international relations and industrial relations.

I participated in several networks of excellence on several aspects large-scale distributed computing systems (from Broadcast, 1992, to the Euro-TM Action on Transactional Memories, current). I created the ERSADS series of winter schools in distributed systems (European Research Symposium on Advanced Distributed Systems, 1995–2001).

4.2 Expert advice

I was a member of the "Consolidator Grants" for the PE6 (Computer Science) selection panel of the European Research Council (ERC), for the period 2012–2016.

I recently co-chaired Dagstuhl workshops, "Consistency in Distributed Systems" (2013) and "Security and Dependability for Federated Cloud Platforms" (2012). I will co-chair the Dagstuhl workshop "Data Consistency in Distributed Systems: Algorithms, Programs, and Databases" in Feb. 2018.

I am a member of the External Advisory Board of the NOVA Laboratory for Computer Science and Informatics (NOVA LINCS) of Universidade Nova de Lisboa.

I was PC co-chair for OPODIS 2014 and I am currently a member of the OPODIS Steering Committee (2014–2016).

I was the Vice-Chair of COST Action IC1001 Euro-TM.

I participated in various program committees, notably ASPLOS 2017, 2016 (external), and 2012, OPODIS 2016, 2013 and 2007, Middleware 2015, PPoPP

2014 (external), 2008, and 2012, EuroSys 2010; Euro-Par (Track chair) 2005, 2008, etc. I have served as reviewer for journals such as: Concurrency and Computation: Practice and Experience (CCPE, Wiley); Distributed Computing (Springer); J. of Computer and System Sciences (JCSS, Elsevier); J. of Parallel and Distributed Computing (JPDC, Elsevier); Science of Computer Programming (SCP, Elsevier); Technique et Science Informatiques (TSI, Hermès); Trans. on Autonomous and Adaptive Systems (TAAS, ACM); Trans. on Computer Systems (TOCS, ACM); Trans. on Computers (IEEE); Trans. on Knowledge and Data Engineering (TKDE, IEEE); Trans. on Parallel and Distributed Systems (TPDS, IEEE); Trans. on Programming Languages and Systems (TOPLAS, ACM); Very Large Database J. (VLDB).

I have acted as project reviewer for a number of grant agencies, including: European Commission (Esprit, FP6, FP7 and FET); European Research Council (ERC); US-Israel Binational Science Foundation; Swedish Research Council; Swiss National Research Council (SNSF); French Agence Nationale de la Recherche (ANR).

4.3 Funding

Here I list my major sources of external funding in the recent period.

I was the co-ordinator (main PI) of the European project SyncFree, which ran from 2013 to 2016 [77]. SyncFree build upon CRDTs and addressed the scientific and practical challenges of massive-scale available distributed systems. Antidote and the CISE tool are outcomes of SyncFree to address the challenge of building synchronization-free (or -reduced) distributed systems. These results are available as open source on GitHub [76]; more information can be found on the AntidoteDB [75] and Lasp [37] websites. The industrial partners of SyncFree were Basho, Trifork, Rovio and Erlang Solutions Ltd. (ESL). The academic partners were Inria, U. Nova de Lisboa (associated with U. do Minho), Tech. U. Kaiserslautern, U. Catholique de Louvain, and Koç U. The total EU funding for SyncFree was 2 669 995 €, of which 557 661 € for Inria.

This line of research continues with the European project LightKone (2017–2019), co-ordinated by Peter Van Roy, with a total funding of 3 570 994 €, of which 395 568 € for Inria. LightKone aims to push the envelope of scalability to general-purpose edge computing.

I am the PI of the ANR project RainbowFS, which started Jan. 2017 and runs for four years. The partners are Inria/UPMC, CNRS-LIG, Télécom ParisSud, and Scality SA. It aims to use and extend the just-right consistency approach

towards building an entreprise-grade, petabyte-scale file system. Its total funding is 919 534 €, of which 359 554 € for Inria.

Previously I was the PI of the ANR ConcoRDanT project, which aimed to develop the understanding and the technology of CRDTs. The partners were Inria, LORIA, and U. Nova de Lisboa. The total funding was 321 742 €, of which 184 192 € for Inria. ConcoRDanT ran from 2010 to 2014.

Before that, I had received research grants from Google and Microsoft Research.

References

[1] V. Abrossimov, M. Rozier, and M. Shapiro. Generic virtual memory management for operating system kernels. In *Symp. on Op. Sys. Principles (SOSP)*, pages 123–136, Litchfield Park AZ, USA, December 1989. Assoc. for Computing Machinery. doi: 10.1145/74850.74863. URL http://doi.acm.org/10.1145/74850.74863.

[2] Marcos K. Aguilera, Leonardo Querzoni, and Marc Shapiro, editors. *Principles of Distributed Systems*, volume 8878 of *Lecture Notes in Comp. Sc.*, Cortina d'Ampezzo, Italy, December 2014. Springer-Verlag. doi: 10.1007/978-3-319-14472-6. URL http://link.springer.com/book/10.1007/978-3-319-14472-6.

[3] Deepthi Devaki Akkoorath, Alejandro Z. Tomsic, Manuel Bravo, Zhongmiao Li, Tyler Crain, Annette Bieniusa, Nuno Preguiça, and Marc Shapiro. Cure: Strong semantics meets high availability and low latency. In *Int. Conf. on Distributed Comp. Sys. (ICDCS)*, pages 405–414, Nara, Japan, June 2016. doi: 10.1109/ICDCS.2016.98. URL http://doi.ieeecomputersociety.org/10.1109/ICDCS.2016.98.

[4] Masoud Saeida Ardekani, Pierre Sutra, and Marc Shapiro. G-DUR: A middleware for assembling, analyzing, and improving transactional protocols. In *Int. Conf. on Middleware (MIDDLEWARE)*, pages 13–24, Bordeaux, France, December 2014. doi: 10.1145/2663165.2663336. URL http://dx.doi.org/10.1145/2663165.2663336.

[5] Valter Balegas, Nuno Preguiça, Rodrigo Rodrigues, Sérgio Duarte, Carla Ferreira, Mahsa Najafzadeh, and Marc Shapiro. Putting the consistency back into eventual consistency. In *W. on Large-Scale Dist. Sys. and Middleware (LADIS)*, Cambridge, UK, April 2014.

[6] Valter Balegas, Sérgio Duarte, Carla Ferreira, Rodrigo Rodrigues, Nuno Preguiça, Mahsa Najafzadeh, and Marc Shapiro. Towards fast invariant preservation in geo-replicated systems. *Operating Systems Review*, 49(1):121–125, January 2015. doi: 10.1145/2723872.2723889. URL http://doi.acm.org/10.1145/2723872.2723889. Selected Papers from LADIS 2014 Eighth Workshop on Large-Scale Distributed Systems and Middleware.

[7] Valter Balegas, Nuno Preguiça, Rodrigo Rodrigues, Sérgio Duarte, Carla Ferreira, Mahsa Najafzadeh, and Marc Shapiro. Putting consistency back into eventual consistency. In *Euro. Conf. on Comp. Sys. (EuroSys)*, pages 6:1–6:16, Bordeaux, France, April 2015. doi: 10.1145/2741948.2741972. URL http://dl.acm.org/citation.cfm?doid=2741948.2741972.

[8] Valter Balegas, Diogo Serra, Sérgio Duarte, Carla Ferreira, Marc Shapiro, Rodrigo Rodrigues, and Nuno Preguiça. Extending eventually consistent cloud databases for enforcing numeric invariants. In *Symp. on Reliable Dist. Sys. (SRDS)*, pages 31–36, Montréal, Canada, September 2015. IEEE Comp. Society, IEEE Comp. Society. doi: 10.1109/SRDS.2015.32. URL http://dx.doi.org/10.1109/SRDS.2015.32.

[9] Valter Balegas, Cheng Li, Mahsa Najafzadeh, Daniel Porto, Allen Clement, Sérgio Duarte, Carla Ferreira, Johannes Gehrke, João Leitão, Nuno Preguiça, Rodrigo Rodrigues, Marc Shapiro, and Viktor Vafeiadis. Geo-replication: Fast if possible, consistent if necessary. *Data Engineering Bulletin*, 39(1):81–92, March 2016. URL http://sites.computer.org/debull/A16mar/A16MAR-CD.pdf.

[10] Lamia Benmouffok, Jean-Michel Busca, Joan Manuel Marquès, Marc Shapiro, Pierre Sutra, and Georgios Tsoukalas. Telex: A semantic platform for cooperative application development. In *Conf. Française sur les Systèmes d'Exploitation (CFSE)*, Toulouse, France, September 2009. URL http://lip6.fr/Marc.Shapiro/papers/Telex-CFSE-2009.pdf.

[11] Annette Bieniusa, Marek Zawirski, Nuno Preguiça, Marc Shapiro, Carlos Baquero, Valter Balegas, and Sérgio Duarte. Brief announcement: Semantics of eventually consistent replicated sets. In Marcos K. Aguilera, editor, *Int. Symp. on Dist. Comp. (DISC)*, volume 7611 of *Lecture Notes in Comp. Sc.*, pages 441–442, Salvador, Bahia, Brazil, October 2012. Springer-Verlag. doi: 10.1007/978-3-642-33651-5_48. URL http://dx.doi.org/10.1007/978-3-642-33651-5_48.

[12] Annette Bieniusa, Marek Zawirski, Nuno Preguiça, Marc Shapiro, Carlos Baquero, Valter Balegas, and Sérgio Duarte. An optimized conflict-free replicated set. Rapport de Recherche RR-8083, Institut National de la Recherche en Informatique et Automatique (Inria), Rocquencourt, France, October 2012. URL http://hal.inria.fr/hal-00738680.

[13] Xavier Blondel, Paulo Ferreira, and Marc Shapiro. Implementing garbage collection in the PerDiS system. In *W. on Persistent Object Sys. (POS)*, Tiburon CA, USA, August 1998. URL http://lip6.fr/Marc.Shapiro/papers/IGCPS_pos8.pdf.

[14] Loïck Briot, Pascal Urso, and Marc Shapiro. High responsiveness for group editing CRDTs. In *Int. Conf. on Supporting Group Work*, pages 51–60, Sanibel Island, FL, USA, November 2016. Assoc. for Computing Machinery, Assoc. for Computing Machinery. doi: 10.1145/2957276.2957300. URL http://dx.doi.org/10.1145/2957276.2957300.

[15] Tyler Crain and Marc Shapiro. Designing a causally consistent protocol for geo-distributed partial replication. In Carlos Baquero and Marco Serafini, editors, *W. on Principles and Practice of Consistency for Distr. Data (PaPoC)*, co-located with EuroSys 2015, Bordeaux, France, April 2015. ACM SIG on Op. Sys. (SIGOPS), Assoc. for Computing Machinery. doi: 10.1145/2745947.2745953. URL http://dx.doi.org/10.1145/2745947.2745953.

[16] Paulo Ferreira and Marc Shapiro. Distribution and persistence in multiple and heterogeneous address spaces. In *Int. W. on Object Orientation in Op. Sys. (IWOOOS)*, pages 83–93, Asheville, NC, USA, December 1993. IEEE Comp. Society. doi: 10.1109/IWOOOS.1993.324924.

[17] Paulo Ferreira and Marc Shapiro. Garbage collection and DSM consistency. In *Symp. on Op. Sys. Design and Implementation (OSDI)*, pages 229–241, Monterey CA, USA, November 1994. ACM. URL http://www.usenix.org/publications/library/proceedings/osdi/ferr.html.

[18] Paulo Ferreira and Marc Shapiro. Garbage collection of persistent objects in distributed shared memory. In *W. on Persistent Object Sys. (POS)*, pages 176–191, Tarascon, France, September 1994. Springer-Verlag. URL http://lip6.fr/Marc.Shapiro/papers/GC-PERS-DSM_POS94.pdf.

[19] Paulo Ferreira and Marc Shapiro. Larchant: Persistence by reachability in distributed shared memory through garbage collection. In *Int. Conf. on Distributed Comp. Sys. (ICDCS)*, pages 394–401, Hong Kong, May 1996. doi: 10.1109/ICDCS.1996.507987. URL http://lip6.fr/Marc.Shapiro/papers/LPRDSMGC_icdcs96.pdf.

[20] Paulo Ferreira and Marc Shapiro. Modelling a distributed cached store for garbage collection: the algorithm and its correctness proof. In Eric Jul, editor, *Euro. Conf. on Object-Oriented Pging. (ECOOP)*, volume 1445 of *Lecture Notes in Comp. Sc.*, pages 234–259, Brussels, Belgium, July 1998. Springer-Verlag. doi: 10.1007/BFb0054094. URL http://dx.doi.org/10.1007/BFb0054094.

[21] Paulo Ferreira, Marc Shapiro, Xavier Blondel, Olivier Fambon, João Garcia, Sytse Kloosterman, Nicolas Richer, Marcus Roberts, Fadi Sandakly, George Coulouris, Jean Dollimore, Paulo Guedes, Daniel Hagimont, and Sacha Krakowiak. PerDiS: design, implementation, and use of a PERsistent DIstributed Store. In S. Krakowiak and S. K. Shrivastava, editors, *Recent Advances in Distributed Systems*, volume 1752 of *Lecture Notes in Comp. Sc.*, chapter 18, pages 427–452. Springer-Verlag, February 2000. doi: 10.1007/3-540-46475-1_18. URL http://www.springerlink.com/content/vmptt4r5k418udvy/.

[22] Lokesh Gidra, Gaël Thomas, Julien Sopena, and Marc Shapiro. Assessing the scalability of garbage collectors on many cores. *Best papers from PLOS'11, SIGOPS Oper. Sys. Review (OSR)*, 45(3):15–19, December 2011. doi: 10.1145/2094091.2094096. URL http://doi.acm.org/10.1145/2094091.2094096.

[23] Lokesh Gidra, Gaël Thomas, Julien Sopena, and Marc Shapiro. A study of the scalability of stop-the-world garbage collectors on multicores. In *Int. Conf. on Archi. Support for Prog. Lang. and Systems (ASPLOS)*, pages 229–240, Houston, TX, USA, March 2013. Assoc. for Computing Machinery. doi: 10.1145/2499368.2451142. URL http://dx.doi.org/10.1145/2499368.2451142.

[24] Lokesh Gidra, Gaël Thomas, Julien Sopena, Marc Shapiro, and Nhan Nguyen. NumaGiC: a garbage collector for big data on big NUMA machines. In *Int. Conf. on Archi. Support for Prog. Lang. and Systems (ASPLOS)*, pages 661–673, Istanbul, Turkey, March 2015. Assoc. for Computing Machinery. doi: 10.1145/2694344.2694361. URL http://dx.doi.org/10.1145/2694344.2694361.

[25] Alexey Gotsman, Hongseok Yang, Carla Ferreira, Mahsa Najafzadeh, and Marc Shapiro. 'Cause I'm strong enough: Reasoning about consistency choices in distributed systems. In *Symp. on Principles of Prog. Lang. (POPL)*, pages 371–384, St. Petersburg, FL, USA, 2016. doi: 10.1145/2837614.2837625. URL http://dx.doi.org/10.1145/2837614.2837625.

[26] Yvon Gourhant and Marc Shapiro. FOG/C++: a fragmented-object generator. In *C++ Conference*, pages 63–74, San Francisco, CA, USA, April 1990. Usenix.

[27] Youssef Hamadi and Marc Shapiro. Pushing log-based reconciliation. *Int. J. on Artif. Intelligence Tools (IJAIT)*, 14(3–4):445–458, June 2005. doi: 10.1142/S0218213005002193. URL http://dx.doi.org/10.1142/S0218213005002193.

[28] Claudia-Lavinia Ignat, Gérald Oster, Pascal Molli, Michèle Cart, Jean Ferrié, Anne-Marie Kermarrec, Pierre Sutra, Marc Shapiro, Lamia Benmouffok, Jean-Michel Busca, and Rachid Guerraoui. A comparison of optimistic approaches to collaborative editing of Wiki pages. In *Int. Conf. on Collaborative Comp.: Networking, Apps. and Worksharing (CollaborateCom)*, number 3, White Plains, NY, USA, November 2007. doi: 10.1109/COLCOM.2007.4553878. URL http://dx.doi.org/10.1109/COLCOM.2007.4553878.

[29] Bettina Kemme, André Schiper, Ganesan Ramalingam, and Marc Shapiro. Dagstuhl seminar review: Consistency in distributed systems. *SIGACT News*, 45(1):67–89, March 2014. ISSN 0163-5700. doi: 10.1145/2596583.2596601. URL http://doi.acm.org/10.1145/2596583.2596601.

[30] Anne-Marie Kermarrec, Antony Rowstron, Marc Shapiro, and Peter Druschel. The IceCube approach to the reconciliation of divergent replicas. In *Symp. on Principles of Dist. Comp. (PODC)*, Newport, RI, USA, August 2001. ACM SIGACT-SIGOPS, ACM Press. doi: 10.1145/383962.384020. URL http://doi.acm.org/10.1145/383962.384020.

[31] Nishith Krishna, Marc Shapiro, and Karthikeyan Bhargavan. Brief announcement: Exploring the consistency problem space. In *Symp. on Principles of Dist. Comp.*

(PODC), pages 168–168, Las Vegas, Nevada, USA, July 2005. ACM SIGACT-SIGOPS. doi: 10.1145/1073814.1073845. URL http://doi.acm.org/10.1145/1073814.1073845.

[32] Fabrice le Fessant, Ian Piumarta, and Marc Shapiro. A detection algorithm for distributed cycles of garbage. In *OOPSLA W. on Garbage Collection and Memory Management*, Atlanta, GA, USA, October 1997. URL http://lip6.fr/Marc.Shapiro/papers/DADCG_gcmm97.pdf.

[33] Fabrice le Fessant, Ian Piumarta, and Marc Shapiro. An implementation of complete, asynchronous, distributed garbage collection. In *Conf. on Prog. Lang. Design and Implementation*, Montreal, Canada, June 1998. ACM SIGPLAN. doi: 10.1145/277650.277715. URL http://doi.acm.org/10.1145/277650.277715.

[34] Mihai Leţia, Nuno Preguiça, and Marc Shapiro. Consistency without concurrency control in large, dynamic systems. *Operating Systems Review*, 44(2):29–34, April 2010. doi: 10.1145/1773912.1773921. URL http://doi.acm.org/10.1145/1773912.1773921.

[35] Mesaac Makpangou and Marc Shapiro. The SOS object-oriented communication service. In *Proc. 9th Int. Conf. on Computer Communication*, Tel Aviv, Israel, October–November 1988.

[36] Mesaac Makpangou, Yvon Gourhant, Jean-Pierre Le Narzul, and Marc Shapiro. Fragmented objects for distributed abstractions. In T. L. Casavant and M. Singhal, editors, *Readings in Distributed Computing Systems*, pages 170–186. IEEE Computer Society Press, July 1994.

[37] Christopher S. Meiklejohn and Peter Van Roy. Lasp, a language for distributed, eventually consistent computations. Website https://lasp-lang.org/, 2016.

[38] Jonathan Michaux, Xavier Blanc, Pierre Sutra, and Marc Shapiro. A semantically rich approach for collaborative model edition. In *Symp. on Applied Computing (SAC)*, volume 26, pages 1470–1475, TaiChung, Taiwan, March 2011. ACM SIGAPP, Assoc. for Computing Machinery. doi: 10.1145/1982185.1982500. URL http://doi.acm.org/10.1145/1982185.1982500.

[39] Mahsa Najafzadeh and Marc Shapiro. Demo of the CISE tool, November 2015. URL https://youtu.be/HJjWqNDh-GA. YouTube video.

[40] Mahsa Najafzadeh, Alexey Gotsman, Hongseok Yang, Carla Ferreira, and Marc Shapiro. The CISE tool: Proving weakly-consistent applications correct. In *W. on Principles and Practice of Consistency for Distr. Data (PaPoC)*, EuroSys 2016 workshops, London, UK, April 2016. ACM SIG on Op. Sys. (SIGOPS), Assoc. for Computing Machinery. doi: 10.1145/2911151.2911160. URL http://dx.doi.org/10.1145/2911151.2911160.

[41] David Plainfossé and Marc Shapiro. A survey of distributed garbage collection techniques. In Henry G. Baker, editor, *Int. W. on Memory Management (IWMM)*, volume 986 of *Lecture Notes in Comp. Sc.*, pages 211–249, Kinross, Scotland, UK, September 1995. Springer-Verlag. doi: 10.1007/3-540-60368-9_26. URL http://lip6.fr/Marc.Shapiro/papers/SDGC_iwmm95.pdf.

[42] Nuno Preguiça, Marc Shapiro, and Caroline Matheson. Semantics-based reconciliation for collaborative and mobile environments. In *Int. Conf. on Coop. Info. Sys. (CoopIS)*, volume 2888 of *Lecture Notes in Comp. Sc.*, pages 38–55, Catania, Sicily, Italy, November 2003. Springer-Verlag. URL http://www.springerlink.com/content/xygj6u96h1kgew05/.

[43] Nuno Preguiça, Joan Manuel Marquès, Marc Shapiro, and Mihai Leţia. A commutative replicated data type for cooperative editing. In *Int. Conf. on Distributed Comp. Sys. (ICDCS)*, pages 395–403, Montréal, Canada, June 2009. doi: 10.1109/ICDCS.2009.20. URL http://doi.ieeecomputersociety.org/10.1109/ICDCS.2009.20.

[44] Masoud Saeida Ardekani, Marek Zawirski, Pierre Sutra, and Marc Shapiro. The space complexity of transactional interactive reads. In *Int. W. on Hot Topics in Cloud Data Processing (HotCDP)*, pages 4:1–4:5, Bern, Switzerland, April 2012. Assoc. for Computing Machinery. doi: 10.1145/2169090.2169094. URL http://doi.acm.org/10.1145/2169090.2169094.

[45] Masoud Saeida Ardekani, Pierre Sutra, and Marc Shapiro. Non-Monotonic Snapshot Isolation: scalable and strong consistency for geo-replicated transactional systems. In *Symp. on Reliable Dist. Sys. (SRDS)*, pages 163–172, Braga, Portugal, October 2013. IEEE Comp. Society. doi: 10.1109/SRDS.2013.25. URL http://dx.doi.org/10.1109/SRDS.2013.25.

[46] Masoud Saeida Ardekani, Pierre Sutra, Marc Shapiro, and Nuno Preguiça. On the scalability of snapshot isolation. In Felix Wolf, Bernd Mohr, and Dieter Mey, editors, *Euro. Conf. on Parallel and Dist. Comp. (Euro-Par)*, volume 8097 of *Lecture Notes in Comp. Sc.*, pages 369–381, Aachen, Germany, August 2013. Springer-Verlag. doi: 10.1007/978-3-642-40047-6_39. URL http://dx.doi.org/10.1007/978-3-642-40047-6_39.

[47] Yasushi Saito and Marc Shapiro. Optimistic replication. *ACM Computing Surveys*, 37(1):42–81, March 2005. doi: 1057977.1057980. URL http://doi.acm.org/10.1145/1057977.1057980.

[48] Marc Shapiro. An experiment in distributed program design, using control enrichment. In *Int. Conf. on Distributed Comp. Sys. (ICDCS)*, Miami-Ft. Lauderdale FL, USA, October 1982.

[49] Marc Shapiro. Structure and encapsulation in distributed systems: the Proxy Principle. In *Int. Conf. on Distributed Comp. Sys. (ICDCS)*, pages 198–204, Cambridge, MA, USA, May 1986. IEEE.

[50] Marc Shapiro. A fault-tolerant, scalable, low-overhead distributed garbage detection protocol. In *Symp. on Reliable Dist. Sys. (SRDS)*, pages 208–217, Pisa, Italy, October 1991. doi: 10.1109/RELDIS.1991.145426.

[51] Marc Shapiro. A binding protocol for distributed shared objects. In *Int. Conf. on Distributed Comp. Sys. (ICDCS)*, pages 134–141, Poznan, Poland, June 1994. doi: 10.1109/ICDCS.1994.302403. URL http://ieeexplore.ieee.org/iel2/980/7460/00302403.pdf?tp=&arnumber=302403&isnumber=7460.

[52] Marc Shapiro. Réplication : les approches optimistes (conf. invitée). In Philippe Pucheral, editor, *Journées Bases de Données Avancées (BDA)*, Évry, France, October 2002. Invited Talk.

[53] Marc Shapiro. Practical proofs of concurrent programs. In *Int. Conf. on Functional Programming (ICFP)*, pages 123–123, Portland, Oregon, USA, September 2006. ACM Sigplan, Assoc. for Computing Machinery. doi: 10.1145/1159803.1159819. URL http://doi.acm.org/10.1145/1159803.1159819. Invited Talk.

[54] Marc Shapiro and Paulo Ferreira. Larchant-RDOSS: a distributed shared persistent memory and its garbage collector. In Jean-Michel Hélary and Michel Raynal, editors, *W. on Distributed Algorithms (WDAG)*, number 972 in Lecture Notes in Comp. Sc., pages 198–214, Le Mont Saint-Michel, France, September 1995. Springer-Verlag. doi: 10.1007/BFb0022148. URL http://lip6.fr/Marc.Shapiro/papers/LRDSPMGC_wdag95.pdf.

[55] Marc Shapiro and Nuno Preguiça. Designing a commutative replicated data type. Rapport de Recherche RR-6320, Institut National de la Recherche en Informatique et Automatique (Inria), Rocquencourt, France, October 2007. URL http://lip6.fr/Marc.Shapiro/papers/Commutative-Replicated-Data-Type-RR-6320_2007-10.pdf.

[56] Marc Shapiro, Philippe Gautron, and Laurence Mosseri. Persistence and migration for C++ objects. In Stephen Cook, editor, *Euro. Conf. on Object-Oriented Pging. (ECOOP)*, British Computer Society Workshop Series, pages 191–204, Nottingham, GB, July 1989. The British Computer Society, Cambridge University Society.

[57] Marc Shapiro, Yvon Gourhant, Sabine Habert, Laurence Mosseri, Michel Ruffin, and Céline Valot. SOS: An object-oriented operating system — assessment and perspectives. *Computing Systems*, 2(4):287–338, December 1989. URL http://www.usenix.org/publications/compsystems/1989/fall.html.

[58] Marc Shapiro, Peter Dickman, and David Plainfossé. Robust, distributed references and acyclic garbage collection. In *Symp. on Principles of Dist. Comp. (PODC)*,

pages 135–146, Vancouver, Canada, August 1992. ACM. doi: 10.1145/135419.135448. URL http://doi.acm.org/10.1145/135419.135448. Superseded by [59]: corrects a bug, more elegant, more informative.

[59] Marc Shapiro, Peter Dickman, and David Plainfossé. SSP chains: Robust, distributed references supporting acyclic garbage collection. Rapport de Recherche 1799, Institut National de la Recherche en Informatique et Automatique (Inria), Rocquencourt, France, November 1992. URL http://lip6.fr/Marc.Shapiro/papers/SSPC_rr1799.pdf.

[60] Marc Shapiro, Fabrice le Fessant, and Paulo Ferreira. Recent advances in distributed garbage collection. In S. Krakowiak and S. K. Shrivastava, editors, *Recent Advances in Distributed Systems*, volume 1752 of *Lecture Notes in Comp. Sc.*, chapter 5, pages 104–126. Springer-Verlag, February 2000. doi: 10.1007/3-540-46475-1_5. URL http://www.springerlink.com/content/11xn227wnvhn0972/.

[61] Marc Shapiro, Karthikeyan Bhargavan, Yek Chong, and Youssef Hamadi. Brief announcement: A formalism for consistency and partial replication. In Rachid Guerraoui, editor, *Int. Symp. on Dist. Comp. (DISC)*, volume 3274/2004 of *Lecture Notes in Comp. Sc.*, Trippenhuis, Amsterdam, the Netherlands, October 2004. ISBN 0302-9743.

[62] Marc Shapiro, Karthikeyan Bhargavan, and Nishith Krishna. A constraint-based formalism for consistency in replicated systems. In *Int. Conf. on Principles of Dist. Sys. (OPODIS)*, number 3544 in Lecture Notes in Comp. Sc., pages 331–345, Grenoble, France, December 2004. doi: 10.1007/11516798_24. URL http://dx.doi.org/10.1007/11516798_24.

[63] Marc Shapiro, Nuno Preguiça, and James O'Brien. Rufis: mobile data sharing using a generic constraint-oriented reconciler. In *Conf. on Mobile Data Management*, pages 146–151, Berkeley, CA, USA, January 2004. doi: 10.1109/MDM.2004.1263052. URL http://lip6.fr/Marc.Shapiro/papers/mdm-2004-final.pdf.

[64] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. A comprehensive study of Convergent and Commutative Replicated Data Types. Rapport de Recherche 7506, Institut National de la Recherche en Informatique et Automatique (Inria), Rocquencourt, France, January 2011.

[65] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-free replicated data types. In Xavier Défago, Franck Petit, and V. Villain, editors, *Int. Symp. on Stabilization, Safety, and Security of Dist. Sys. (SSS)*, volume 6976 of *Lecture Notes in Comp. Sc.*, pages 386–400, Grenoble, France, October 2011. Springer-Verlag. doi: 10.1007/978-3-642-24550-3_29. URL http://www.springerlink.com/content/3rg39l2287330370/.

[66] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Convergent and commutative replicated data types. *Bulletin of the European Association for Theoretical Computer Science (EATCS)*, (104):67–88, June 2011. URL http://www.eatcs.org/images/bulletin/beatcs104.pdf.

[67] Marc Shapiro, Masoud Saeida-Ardekani, and Pierre Sutra. Exploring the spectrum of strongly-consistent transactional protocols. In *W. on Large-Scale Dist. Sys. and Middleware (LADIS)*, Cambridge, UK, October 2014. URL http://ladisworkshop.org/node/10.

[68] Marc Shapiro, Masoud Saeida Ardekani, and Gustavo Petri. Consistency in 3D. In Josée Desharnais and Radha Jagadeesan, editors, *Int. Conf. on Concurrency Theory (CONCUR)*, volume 59 of *Leibniz Int. Proc. in Informatics (LIPICS)*, pages 3:1–3:14, Québec, Québec, Canada, August 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany. doi: 10.4230/LIPIcs.CONCUR.2016.3. URL http://drops.dagstuhl.de/opus/volltexte/2016/6188/pdf/LIPIcs-CONCUR-2016-3.pdf.

[69] Pierre Sutra and Marc Shapiro. Comparing optimistic database replication techniques. In *Bases de Données Avancées (BDA)*, Marseille, France, October 2007. URL http://lip6.fr/Marc.Shapiro/papers/sutra-shapiro-bda2007.pdf.

[70] Pierre Sutra and Marc Shapiro. Fault-tolerant partial replication in large-scale database systems. In *Euro. Conf. on Parallel and Dist. Comp. (Euro-Par)*, pages 404–413, Las Palmas de Gran Canaria, Spain, August 2008. doi: 10.1007/978-3-540-85451-7_44. URL http://www.springerlink.com/content/g667712123656274/.

[71] Pierre Sutra and Marc Shapiro. Fast Genuine Generalized Consensus. Rapport de Recherche ???, Institut National de la Recherche en Informatique et Automatique (Inria), Rocquencourt, France, May 2010. URL http://hal.archives-ouvertes.fr/hal-00476885.

[72] Pierre Sutra and Marc Shapiro. Fast Genuine Generalized Consensus. In *Symp. on Reliable Dist. Sys. (SRDS)*, pages 255–264, Madrid, Spain, October 2011. doi: 10.1109/SRDS.2011.38. URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6076784.

[73] Pierre Sutra, João Barreto, and Marc Shapiro. Decentralised commitment for optimistic semantic replication. In *Int. Conf. on Coop. Info. Sys. (CoopIS)*, Vilamoura, Algarve, Portugal, November 2007. doi: 10.1007/978-3-540-76848-7_21. URL http://www.springerlink.com/content/u126126582647jmx/.

[74] Vinh Tao, Marc Shapiro, and Vianney Rancurel. Merging semantics for conflict updates in geo-distributed file systems. In *ACM Int. Systems and Storage Conf.*

(Systor), pages 10.1–10.12, Haifa, Israel, May 2015. doi: 10.1145/2757667.2757683. URL http://dx.doi.org/10.1145/2757667.2757683.

[75] The SyncFree Consortium. AntidoteDB: A planet-scale, available, transactional database with strong semantics. Website http://antidoteDB.eu/, .

[76] The SyncFree Consortium. SyncFree GitHub. URL https://github.com/SyncFree, .

[77] The SyncFree Consortium. The SyncFree European FP7 project. Website http://syncfree.lip6.fr/, .

[78] Alejandro Z. Tomsic, Tyler Crain, and Marc Shapiro. An empirical perspective on causal consistency. In Carlos Baquero and Marco Serafini, editors, *W. on Principles and Practice of Consistency for Distr. Data (PaPoC)*, co-located with EuroSys 2015, Bordeaux, France, April 2015. ACM SIG on Op. Sys. (SIGOPS), Assoc. for Computing Machinery. doi: 10.1145/2745947.2745949. URL http://dx.doi.org/10.1145/2745947.2745949.

[79] Alejandro Z. Tomsic, Tyler Crain, and Marc Shapiro. PhysiCS-NMSI: efficient consistent snapshots for scalable snapshot isolation. In *W. on Principles and Practice of Consistency for Distr. Data (PaPoC)*, London, UK, April 2016. Euro. Conf. on Comp. Sys. (EuroSys), Assoc. for Computing Machinery. doi: 10.1145/2911151.2911166. URL http://dx.doi.org/10.1145/2911151.2911166.

[80] Viktor Vafeiadis, Maurice Herlihy, Tony Hoare, and Marc Shapiro. Proving correctness of highly-concurrent linearisable objects. In *Symp. on Principles and Practice of Parallel Prog. (PPoPP)*, pages 129–136, New York, USA, March 2006. doi: 10.1145/1122971.1122992. URL http://doi.acm.org/10.1145/1122971.1122992.

[81] Marek Zawirski, Marc Shapiro, and Nuno Preguiça. Asynchronous rebalancing of a replicated tree. In *Conf. Française sur les Systèmes d'Exploitation (CFSE)*, page 12, Saint-Malo, France, May 2011. URL http://lip6.fr/Marc.Shapiro/papers/Asynch%20rebalancing%20of%20a%20replicated%20tree%20Zawirski-CFSE-2011.pdf.

[82] Marek Zawirski, Nuno Preguiça, Sérgio Duarte, Annette Bieniusa, Valter Balegas, and Marc Shapiro. Write fast, read in the past: Causal consistency for client-side applications. In *Int. Conf. on Middleware (MIDDLEWARE)*, pages 75–87, Vancouver, BC, Canada, December 2015. ACM/IFIP/Usenix. doi: 10.1145/2814576.2814733. URL http://dx.doi.org/10.1145/2814576.2814733.

[83] Marek Zawirski, Carlos Baquero, Annette Bieniusa, Nuno Preguiça, and Marc Shapiro. Eventually consistent register revisited. In *W. on Principles and Practice of Consistency for Distr. Data (PaPoC)*, London, UK, April 2016. Euro. Conf. on Comp. Sys. (EuroSys), Assoc. for Computing Machinery. doi: 10.1145/2911151.2911157. URL http://dx.doi.org/10.1145/2911151.2911157.