A formalism for consistency and partial replication

Marc Shapiro, Karthikeyan Bhargavan, Yek Chong, Youssef Hamadi

A formalism for consistency and partial replication

Marc Shapiro, Karthikeyan Bhargavan, Yek Chong,[1] Youssef Hamadi
Microsoft Research Cambridge

21 June 2004

Technical Report
MSR-TR-2004-58

Replication protocols are complex and it is difficult to compare their consistency properties. To this effect, we propose a formalism where a replica executes *actions* subject to *constraints* in its local view or *multilog*. Schedules are selected non-deterministically from the set of sound schedules. This set grows with the number of actions and shrinks as the number of constraints increases. If the size of the set is one, the site has converged; if the size becomes zero, the site has detected an unrecoverable conflict. If every site runs the same schedule, they are consistent. We formalise this intuitive concept of consistency in four different ways, which generalise classical consistency criteria and which expose different aspects of consistency protocols. We prove them equivalent. We provide the first formal definition of consistency for partially replicated data. In general, achieving consistency entails global consensus; we exhibit sufficient conditions for deciding locally and derive a new decentralised protocol. In a separate technical report we prove the consistency of some published consistency protocols; this underscores the deep commonalities between them.

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
http://www.research.microsoft.com

[1]University of Cambrige Computer Laboratory

1 Introduction

Replicating data in a distributed system improves availability at the cost of maintaining consistency, since each site's view may be partial or stale. Although a number of protocols have been proposed to achieve various degrees of consistency [14], we lack a common framework for understanding and comparing them. This paper presents such a framework.

In our model, a replicated system consists of a database, replicated at several sites, and a replication protocol that reacts to an environment consisting of users and the underlying network. Users issue *actions* to query or update the database. Each site receives these actions in arbitrary order and executes a schedule, a sequence of received actions. The chosen schedule completely determines the state of the replica. Not all schedules are sound, however, since an actions may conflict with another, or causally depend on another, or users may require some actions to occur before others. These relationships between actions can be represented as binary constraints. We introduce a simple language of constraints between actions. These constraints are adequate to encode many kinds of user requirements, application semantics, and protocol decisions.

A replication protocol takes as input a set of actions and constraints, and ensures that, eventually, the schedules executed at all sites satisfy the constraints and are equivalent. This is the well-known notion of eventual consistency. However, eventual consistency says nothing about the intermediate schedules at the replicas. For instance, it allows a site to execute an intermediate schedule that violates some user constraints. Instead, we want the users to be able to read meaningful data from each replica even before the final decision is made.

Hence, we identify three new notions of consistency for replication protocols. The Uniform Local Soundness property ensures only that every intermediate schedule executed at a site satisfies the user constraints. The Mergeability property guarantees that the decisions taken by different sites do not contradict each other. The Common Monotonic Prefix Property guarantees that the schedules picked by sites over time have a monotonically growing prefix. This prefix represents actions that can be safely committed (and even deleted) as they will never be rolled back.

We compare these four different formulations of consistency. We show that under certain liveness conditions, they are all equivalent. These properties guarantee that the intermediate decisions made in a replication protocol are sound, consistent, and stable. While our definitions are for the simple constraint language, these notions are quite general.

We extend the Mergeability conditions to encompass partial replication.

Our formulations clarify that in the general case, consistency entails consensus; however we study sufficient conditions for consistency that can be evaluated locally, and describe a new protocol based on this insight.

The paper proceeds as follows. Section 2 overviews the basic formalism. Section 3 defines and compares consistency properties. We examine partial replication Section 4. Section 5 compares with related work, and we conclude in Section 6 with a summary of contributions and future work.

A separate technical report [15] provides a complete formal treatment. Here we focus on presenting the intuitions and keep the formalism to a minimum.

2 Basic formalism

Each site in a replicated system maintains a local view called *multilog*.[1] The current state results from executing a *sound* (i.e., valid) schedule computed from the multilog. Over time the multilog grows (and conceptually never shrinks) by addition of actions and constraints, either *submitted* by local clients or received from remote sites. The set of sound schedules grows with the number of actions and shrinks as the number of constraints increases.

2.1 Actions and schedules

Slightly more formally, A is the (finite) set of actions $\text{INIT}, \alpha, \beta, \ldots$. Actions are assumed deterministic but are otherwise uninterpreted. The *non-action* $\overline{\alpha}$ is a placeholder with no effect (non-actions will be useful when discussing liveness). Action INIT represents the initial state and has no effect.

A *schedule* is a non-empty sequence of actions and non-actions, for instance $S = \text{INIT}.\alpha.\overline{\beta}.\gamma$. In this example, α is *executed* (noted $\alpha \in S$), and β is non-executed (noted $\overline{\beta} \in S$); all four actions are said *scheduled* (noted $sched(\alpha, S)$). A given action may appear only once in a schedule, either as executed or as non-executed. The ordering is noted $<_S$. Every schedule starts with INIT.

Actions commute unless specified otherwise by the notation $\alpha \leftrightarrow \beta$ (read "non-commuting"). A non-action commutes with every action and non-action. Two schedules are *equivalent* ($S_1 \equiv S_2$) if they execute the same actions, and non-commuting actions execute in the same order.

Commutativity allows us to model a number of real-world cases of schedule equivalence: (i) Classically, actions commute if both are reads, or if they access independent variables. (ii) Overwriting: in some systems an out-of-order write has no effect; then writes effectively commute. For instance in timestamped replication (Last Writer Wins) [14], writing a variable tests whether the write timestamp is greater than the object's; if so the write takes effect, otherwise it is a no-op [15]. (iii) Reconciliation: for instance in Operational Transformation [18, 20], two actions submitted concurrently execute in arbitrary order. The second one to execute is transformed to ignore the effect of the first, in effect rendering them commutative. (iv) Failure or aborts: An action that fails or aborts becomes *dead*, i.e., appears as a non-action in all schedules, which commutes with all actions.

2.2 Multilogs and sound schedules

Multilog $M = (K, \rightarrow, \triangleright)$ represents a site's view. K is the set of known actions

[1] We call it a multilog and not a log, because it contains actions submitted at several sites and the actions are not ordered.

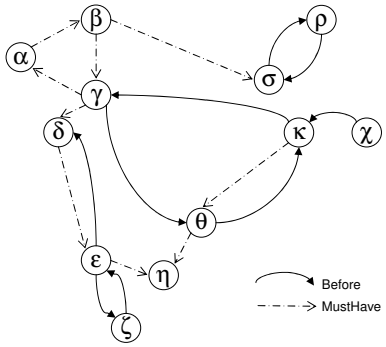

Figure 1: Example constraints. α, β and γ form a *parcel*, an atomic (i.e., all-or-nothing) execution. γ executes only if δ also executes. δ is *causally dependent* on ϵ. ϵ and ζ *mutually exclude* each other. Only two actions out of the three γ, θ and κ can execute. If both χ and κ execute, χ comes first.

($K \subseteq A$); $\rightarrow$ and $\triangleright$ are the set of known constraints. The relation $\rightarrow \subseteq A \times A$ (pronounced Before) is not acyclic, nor reflexive, nor transitive. Relation $\triangleright \subseteq A \times A$ (pronounced MustHave) is transitive and reflexive. By convention, for any action $\alpha \in A$, $\text{INIT} \rightarrow \alpha$ and $\alpha \triangleright \text{INIT}$; this is left implicit in the rest of the paper.

The set of sound schedules of M is noted $\Sigma(M)$; M is said sound if $\Sigma(M) \neq \emptyset$. Schedule $S \in \Sigma(M)$ iff:

- S contains all actions in K: $\alpha \in K \Rightarrow \mathit{sched}(\alpha, S)$.
- Actions that execute in S are ordered by $\rightarrow$: $\alpha, \beta \in S \wedge \alpha \rightarrow \beta \Rightarrow \alpha <_S \beta$.
- MustHave behaves like implication: $\alpha \in S \wedge \alpha \triangleright \beta \Rightarrow \beta \in S$.

For instance, the multilogs $M1 = (\{\alpha\}, \emptyset, \{\text{INIT} \triangleright \alpha\})$ and $M2 = (\{\alpha\}, \{\alpha \rightarrow \alpha\}, \emptyset)$ are both sound. Their sound schedules are $\Sigma(M1) = \{\text{INIT}.\alpha\}$ and $\Sigma(M2) = \{\text{INIT}\}$. Their union $M3 = (\{\alpha\}, \{\alpha \rightarrow \alpha\}, \{\text{INIT} \triangleright \alpha\})$ is not sound. Although it contains a $\rightarrow$ cycle, multilog $M4 = (\{\alpha, \beta\}, \{\alpha \rightarrow \beta, \beta \rightarrow \alpha\}, \emptyset)$ is sound, since $\Sigma(M4) = \{\text{INIT}, \text{INIT}.\alpha, \text{INIT}.\beta\}$.

This limited constraints language is surprisingly expressive. We have used it to express the semantics of applications as diverse as a shared calendar, a travel reservation system and a replicated file system [11, 16]. For instance if α creates a directory and β a file in that same directory, the file system submits $\beta \triangleright \alpha \wedge \alpha \rightarrow \beta$ (causal dependence) along with β.

Constraints are also the language for enforcing some properties of replication protocols. For instance, $\text{INIT} \triangleright \alpha$ ensures that α must execute, and $\beta \rightarrow \beta$ ensures β may not execute.

Two multilogs are equivalent if they generate the same set of sound schedules. $M_1 \equiv M_2$ iff $\Sigma(M_1) = \Sigma(M_2)$. Hereafter we identify a multilog with its its equivalence class.

2.3 Significant subsets and events of a replication protocol

Execution strategies vary widely between replication protocols: in some, actions execute immediately, in others they are deferred; execution order may be pre-established or computed; actions may roll back. However a protocol would be useless if it did not reach some final decision for every action. We represent decisions as constraints; the following *significant subsets* capture the possible stages of irrevocable decision:

- **Guaranteed** actions execute in every schedule. $Guar(M)$ is the smallest set satisfying: (1) INIT $\in Guar(M)$. (2) $\forall \beta \in A$: If $\alpha \in Guar(M)$ and $\alpha \rhd \beta$ then $\beta \in Guar(M)$.
- **Dead** actions non-execute in every schedule. $Dead(M)$ is the smallest set satisfying: (1) $\forall \alpha \in A$: If $\beta_1, \ldots, \beta_m \in Guar(M)$, where m is any natural integer, and $\alpha \rightarrow \beta_1 \rightarrow \ldots \rightarrow \beta_m \rightarrow \alpha$, then $\alpha \in Dead(M)$. (2) $\forall \alpha \in A$: If $\beta \in Dead(M)$ and $\alpha \rhd \beta$, then $\alpha \in Dead(M)$.
- A **serialised** action is one that is ordered with respect to all non-commuting actions that execute. $Serialised(M) \stackrel{\text{def}}{=} \{\alpha \in A | \forall \beta \in A, \alpha \leftrightarrow \beta \Rightarrow \alpha \rightarrow \beta \vee \beta \rightarrow \alpha \vee \beta \in Dead(M)\}$
- An action is **decided** once it is either dead, or both guaranteed and serialised. $Decided(M) \stackrel{\text{def}}{=} Dead(M) \cup (Guar(M) \cap Serialised(M))$
- An action is **stable** when its effects cannot change, i.e., it is either dead, or it is guaranteed and serialised and all preceding actions are themselves stable. (In practice, stable actions can be pruned from multilogs.) $Stable(M)$ is the smallest set satisfying: (1) INIT $\in Stable(M)$, (2) $Dead(M) \subseteq Stable(M)$, (3) If $(\alpha \in Guar(M) \cap Serialised(M)) \wedge (\forall \beta \in A : \beta \rightarrow \alpha \Rightarrow \beta \in Stable(M))$ then $\alpha \in Stable(M)$.

Note that if M is sound, every guaranteed action must be known: $Guar(M) \subseteq K$. Also note that $\alpha \rightarrow \alpha \Rightarrow \alpha \in Dead(M)$.

Theorem 1 *M is a sound multilog if and only if no action is both guaranteed and dead: $\Sigma(M) \neq \emptyset \Leftrightarrow Guar(M) \cap Dead(M) = \emptyset$.*

Lemma 1 *If the relation $\rightarrow$ is acyclic, then $Dead(M) = \emptyset$ and M is sound.*

3 Replication and consistency

We turn to the study of replication. Each site i has its own view $M_i(t) = (K_i, \rightarrow_i, \rhd_i)(t)$, evolving over time t.[2]

3.1 Transition rules and site schedules

Every protocol must obey the following *transition rule*.

Transition Rule 1 (Universal transition rule) *$M_i(t)$ may evolve, between time t to $t+1$, only according to the following legal transitions:*

[2] For simplicity we assume discrete time and use a global time notation. The theory does not assume that a site can observe the global time.

1. *Unchanged:* $M_i(t+1) = M_i(t)$
2. *A local client* submits *new actions or constraints, which are added to* $M_i(t+1)$.
3. *Site* receives *actions or constraints from a remote site multilog, which are added to* $M_i(t+1)$.

The two last bullets are not always distinguishable, but we will need to separate them when formalising the eventual consistency property (Section 3.4).

Multilogs are monotonically non-shrinking, which implies that the significant subsets of Section 2.3 are non-shrinking, and that an unsound multilog remains unsound forever.

The current state of site i is the result of running a *site schedule* $S_i(t) \in \Sigma(M_i(t))$. If $|\Sigma(M_i(t))| > 1$ the choice between sound schedules is arbitrary. For instance, for $M_1(1) = (\{\alpha, \beta\}, \emptyset, \emptyset)$ any of the sound schedules INIT, INIT.α, INIT.β, INIT.α.β and INIT.β.α is a valid choice for $S_1(1)$. Adding constraints $\alpha \rightarrow \beta$ and $\beta \rhd \alpha$ at time 2, $S_1(2)$ can be any of $\{$INIT, INIT.α, INIT.α.$\beta\}$. If at time 3 the constraint INIT $\rhd \beta$ is added, the only possibility for $S_1(3)$ is INIT.α.β. At this point, it would be unsound to add $\beta \rightarrow \alpha$.

A specific protocol may have additional transition rules. As an example, let us encode a linearisable protocol [9], i.e., where an action takes effect at some instant in time, and actions execute in taking-effect order. Let us identify the time an action is submitted with when it takes effect. The following transition rule (which assumes a global clock) orders previously-submitted actions before it, and unsubmitted actions afterwards. "Only one action can be submitted per value of t. If α is submitted at time t, then for any action $\beta \neq \alpha$: if $\beta \in \bigcup_j K_j(t-1)$ then $\beta \rightarrow \alpha$, otherwise $\alpha \rightarrow \beta$."

A replicated system based on pessimistic concurrency control, or *pessimistic system*, has transition rules that ensure that at every site and every time $S_i(t)$ is a prefix of $S_i(t+1)$. Otherwise the system is said *optimistic*.

3.2 Eventual decision

As mentioned earlier, useful protocols make irrevocable decisions for every action. We formalise this as a liveness property.

Property 1 (Eventual Decision) *A replicated system has the Eventual Decision (ED) property iff every submitted action is eventually decided:* $\alpha \in K_i(t) \Rightarrow \exists t' : \alpha \in \mathit{Decided}(M_i(t'))$.

Note that the ED property is local to site i. ED does not preclude the trivial implementation that makes every action dead; this would be difficult to rule out formally, since actions might fail.

ED implies that every action eventually becomes stable [15].

3.3 Common Monotonic Strong Prefix (CMSP)

Intuitively, if all sites execute the same schedule then the system is consistent [10]. But this does not work for optimistic protocols where $S_i(t)$ is not nec-

essarily a prefix of $S_i(t+1)$. However, even in optimistic systems, we expect all schedules to have a same (up to equivalence) common stable prefix. We formalise this concept as our first definition of consistency.

We say that a schedule P is a prefix of schedule S, written $P \ll S$, if $S \equiv S'$ where S' is a schedule of the form $P.Q$ for some sequence of actions Q.

Property 2 *A replicated system $M_i(t)$ (i varying over sites, t over time) satisfies the CMSP Property if there exists a function $\pi(i,t)$ such that:*

1. *π is a prefix of* all *sound schedules: $S \in \Sigma(M_i(t)) \Rightarrow \pi(i,t) \ll S$.*
2. *The prefix is equivalent at all sites: $\pi(i,t) \equiv \pi(j,t)$*
3. *The prefix is monotonically non-shrinking over time: $t < t' \implies \pi(i,t) \ll \pi(i,t')$*
4. *Every known action eventually reaches the prefix: $\alpha \in K_i(t) \implies \exists t' : sched(\alpha, \pi(i,t'))$*

We show that the actions in a monotonic strong prefix are stable, and that stable actions form a strong monotonic prefix [15].

3.4 Eventual consistency

Another classical property is Eventual Consistency, which has been used to argue informally about the correctness of optimistic systems such as Grapevine [2] or Bayou [19]. This will be our second definition of consistency.

Property 3 *A system is Eventually Consistent if, if every client stops submitting, and submitted actions are decided, then eventually every site will reach the same sound final value:*

$$\exists T : \forall i, t > T \Rightarrow \textit{The transition of Bullet 2 of Rule 1 does not fire at } i$$
$$\implies$$
$$\exists T', \forall t', t'', i, j : t' > T' \wedge t'' > T' \wedge S_i(t') \in \Sigma(M_i(t')) \wedge S_j(t'') \in \Sigma(M_j(t''))$$
$$\Rightarrow S_i(t') \equiv S_j(t'')$$

The two definitions are equivalent:

Theorem 2 *If every client stops submitting, then a replicated system that satisfies the Eventual Consistency property if and only if it satisfies the CMSP property.*

Since this is a central result, we provide a sketch of the proof.
EC $\Rightarrow$ CMSP: Assume Eventual Consistency (EC); if clients stop submitting, by some time T, the final state at site i is $S_i(T) \in \Sigma(M_i(t))$. By EC, for all i, j, $t', t'' > T$, $S_i(t') \equiv S_j(t'')$. The following function satisfies CMSP: $\pi(i,t) = \begin{cases} \text{INIT} & \text{for } t < T \\ S_i(T) & \text{for } t \geq T \end{cases}$
CMSP $\Rightarrow$ EC: Assume the CMSP Property. Every action is eventually stable, hence eventually decided. Suppose that the client at site i stops submitting at time t_i. By time $T_0 = \max_i(t_i)$ all clients have stopped submitting. Let

$A' = \bigcup_j K_j(T_0)$. According to the CMSP Property, there is a time where every action in A' is in the prefix: $\forall \alpha \in A', \exists t'_i : sched(\alpha, \pi(i, t'_i))$. Assume sound site-multilogs; there exists sound site-schedules. Since no new actions are submitted, A' does not change, and the prefix covers the whole site-schedule: $S_i(t'_i) \equiv \pi(i, t'_i)$. According CMSP, all prefixes are equivalent; by time $T' = \max_k t'_k$ the prefix covers the whole site-schedule at every site; hence the site-schedules are mutually equivalent: $\forall i, j : t' > T', t'' > T' \Rightarrow S_i(t') \equiv \pi(i, t') \equiv \pi(j, t'') \equiv S_j(t'')$. **QED**.

3.5 Mergeability

The Mergeability property captures the intuition that sites should not contradict one another. It says that a hypothetical omniscient observer that could see all sites over all time would not see anything wrong. This is our third definition of consistency.

Property 4 (Mergeability) *A system has the Mergeability property if, given any arbitrary collection of sites $i, i', i'' \ldots$ and any arbitrary collection of times $t, t', t'' \ldots : M_i(t) \cup M_{i'}(t') \cup M_{i''}(t'') \ldots$ is sound.*

Sites may not (even at different times) take mutually unsound decisions. This is easy to ensure in a centralised system, but not in a distributed one. For instance, consider Site 1 has multilog $(\{\alpha\}, \emptyset, \{\text{INIT} \rhd \alpha\})$ and Site 2 has multilog $(\{\alpha\}, \{\alpha \rightarrow \alpha\}, \emptyset)$. They are both sound but not mergeable, as their union $(\{\alpha\}, \{\alpha \rightarrow \alpha\}, \{\text{INIT} \rhd \alpha\})$ is not sound.

Mergeability is equivalent to the two previous definitions.

Theorem 3 *If every client stops submitting, then a replicated system satisfies Eventual Consistency if and only if it satisfies the Mergeability, Eventual Decision and Eventual Propagation properties.*

The proof is omitted; we refer the interested reader to our technical report [15], which also shows that mergeability subsumes serialisability.

3.6 Uniform local soundness

If we strengthen the liveness condition so that every submitted action is eventually received everywhere, then every site becomes an omniscient observer. Then Mergeability reduces to the following Uniform Local Soundness (ULS) property, i.e., multilogs are sound at all times: $\forall i, t : \Sigma(M_i(t)) \neq \emptyset$. This is our fourth definition of consistency.

Theorem 4 *Assuming the Eventual Decision property and that every action submitted at some site is eventually received at all other sites, a system satisfies Uniform Local Soundness and Eventual Constraint Propagation if and only if it satisfies Mergeability.*

In conclusion, our four different formulations of consistency are equivalent (under strong liveness conditions).

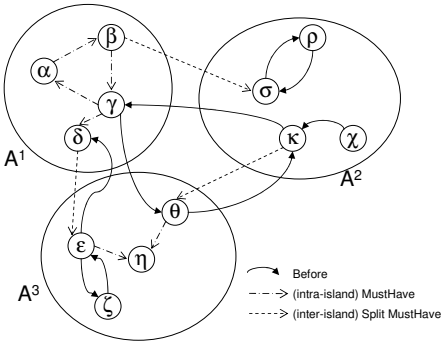

Figure 2: The system of Figure 1 partitioned into three islands.

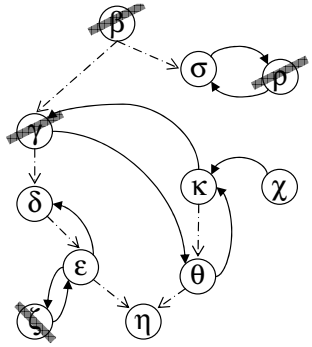

Figure 3: An execution of the Chong-Hamadi algorithm, proceeding from η upwards: greyed actions are dead, the others guaranteed. (α deleted from Figure 1 and layout rearranged.)

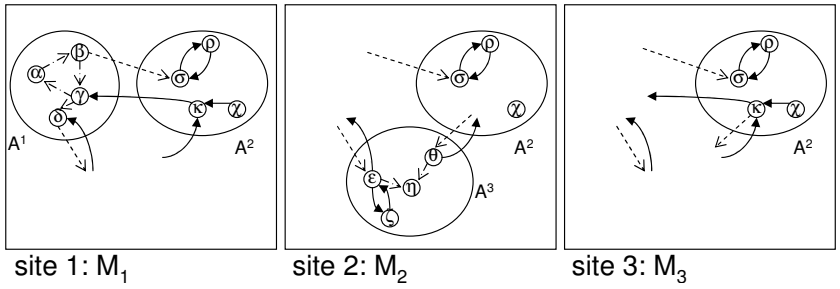

Figure 4: Partial replication of the system of Figure 2. Site 1 replicates islands A^1 and A^2. Site 2 replicates islands A^2 and A^3 but has not yet received action κ. Site 3 only replicates A^2. None of the sites has yet received constraint $\gamma \rightarrow \theta$.

3.7 Sufficient safety conditions

Consistency imposes agreement between all sites, which in the general case entails a consensus. Mergeability would require a consensus "vertically" between sites to to make an action dead or guaranteed. Soundness would require consensus "horizontally" between actions for consistency with their constraints. Yet some practical protocols manage without this complexity; how is this, and can their approach be generalised?

Consider "last-writer wins" timestamped replication. We argued in Section 2.1 that all its actions commute. The $\rightarrow$ graph is empty, hence vacuously acyclic, hence multilogs are sound. Every action is guaranteed, hence decided, and is stable immediately when submitted. This explains why consensus is not needed.

The ESDS protocol [7] assumes that $\rightarrow$ is acyclic and that every action is guaranteed. However, actions do not commute, and ESDS requires a distributed consensus for serialisation.

Many systems centralise consensus at a primary site. Bayou [19] partitions

the data into independent databases, each with its own primary site. Primaries implement consensus for their own actions. Between databases, Bayou assumes commutative actions and absence of constraints, so there is no need for horizontal consensus between different primaries.

If the constraint graph has some well-behaved properties, the horizontal decisions can be safely decentralised; later we will derive an efficient decision protocol from the following observations. Consider for instance an action α that is involved in a single constraint $\alpha \rhd \beta$: then it is always safe to make α dead, regardless of the decision for β. Conversely, if α is only involved in $\gamma \rhd \alpha$, it is always safe to make α guaranteed, regardless of γ. This can be generalised to any acyclic $\rhd$ graph. Taking the example of a chain $\alpha_1 \rhd \ldots \rhd \alpha_n$ it is safe to either: make α_1 dead, then move on to α_2, left to right; or make α_n guaranteed, then move on to α_{n-1}, right to left (it is better to guarantee, so the right-left direction is generally preferable).

The decision regarding each α_i must consider $\rightarrow$ constraints. If α_i is not part of a $\rightarrow$ cycle, the decision may be either guarantee or make dead (although guaranteeing is preferable). If it is part of a $\rightarrow$ cycle, and all other actions in the cycle are guaranteed, the only sound decision is to make α_i dead; otherwise either decision is allowed.

Such local decisions may be sub-optimal. To ensure optimality, viz., that the smallest possible number of actions is made dead, it is necessary to consider the whole graph as in IceCube [11].

4 Partial replication

Up to now we assumed that all data is replicated at every site. Let us now consider partial replication: shared data is partitioned into n disjoint *databases* $D^1, \ldots, D^n$, and we allow a site to replicate an arbitrary subset of the databases (as long as every database is present on at least one site).

Note A^i the set of actions operating on database D^i. Let us assume that each action operates on a single database;[3] then actions are partitioned by $A = A^1 \uplus \ldots \uplus A^n$. A site replicating D^i should receive submitted actions that are in A^i, and the constraints that involve such actions. It does not need to receive actions or constraints for databases it does not replicate. Such a system obviously violates the Common Monotonic Strong Prefix, Eventual Consistency and Uniform Local Soundness properties.

However, Mergeability remains meaningful. Its intuitive meaning is that a hypothetical omniscient observer would see that sites do not contradict each other. Eventual Decision ensures that progress is made.

Property 5 (Consistency in the presence of partial replication) *A distributed system with partial replication is consistent iff it satisfies the Mergeability and Eventual Decision properties.*

[3] An application that operates on several databases submits multiple actions connected by appropriate constraints; see the formalisation of transactions in our technical report [15].

Unfortunately $\rhd$ is not adequate for partial replication, because if $\alpha \rhd \beta$, then a site that executes α must also know β. Therefore we define a version that is "remotable" across islands, Split MustHave, noted $\rhd\!\!\!\rhd \subseteq A \times A$ as follows. For any two actions $\alpha \in A^d$ and $\beta \in A^{d'}$, and for any site i and time t:

$$\alpha \rhd\!\!\!\rhd \beta \stackrel{\text{def}}{=} \begin{cases} \text{If } \{d, d'\} \subseteq \mathit{Islands}_i : \forall S \in \Sigma(M_i(t)), \alpha \in S \Rightarrow \beta \in S \\ \text{Otherwise: } \exists i' : \alpha \in \mathit{Guar}(M_i(t)) \Rightarrow \beta \in \mathit{Guar}(M_{i'}(t)) \end{cases}$$

Practically, site i may guarantee α after it receives a message from i' that β is guaranteed.

All the results seen so far remain true when replacing $\rhd$ with $\rhd\!\!\!\rhd$.

4.1 A decentralised decision algorithm

It is desirable to reach decisions for each database independently, as in Bayou, but nonetheless allow constraints across islands, for instance multi-database parcels. To simplify we assume that vertical consensus is centralised at a primary for each database. We wish to decentralise the horizontal agreement. We derive a new, practical algorithm for this from the sufficient conditions of Section 3.7. The algorithm, previously unpublished, is due to Chong and Hamadi [3].

We make some assumptions about actions and the constraint graph. Because database partitions are independent, any $\leftrightarrow$ relation is internal to an island. In practice, constraints are mainly due to the standard relations shown in Figure 1: parcel, causal dependence or mutual exclusion. Causal dependence is acyclic by construction. Parcels across islands will cause problems, but we assume they are rare. Similarly, inter-island $\rightarrow$ cycles are rare.

Given the above assumptions, it is likely that the $\rhd\!\!\!\rhd$ graph is acyclic. Then we can translate the procedure of Section 3.7 into the following algorithm, illustrated in Figure 3. Given any chain $\alpha_n \rhd\!\!\!\rhd \ldots \rhd\!\!\!\rhd \alpha_1 \rhd\!\!\!\rhd \alpha_0$, the algorithm starts from the maximal element α_0 and works back to the minimal element.[4] Given some arbitrary pair $\alpha \rhd\!\!\!\rhd \beta$ where both α and β are undecided, the primary for β decides first. If β is part of a cycle of $\rightarrow$ and all other actions in the cycle are guaranteed, then make it dead, otherwise guarantee it. The decision is sent to the primary of α; if β is dead then α is made dead. Otherwise, decide for α according to $\rightarrow$ cycles and to intra-island constraints. The algorithm continues as long as there remains an undecided target of a $\rhd\!\!\!\rhd$. Guaranteed actions related by $\rightarrow$ execute in that order, otherwise order is arbitrary. The algorithm terminates and maintains soundness, since it makes only safe unilateral decisions. Information flows in the opposite direction of the $\rhd\!\!\!\rhd$ graph, and no consensus is needed to decide.

In some applications, a $\rightarrow$ cycle may occasionally span islands. Then, the implementation must be careful to avoid concurrently guaranteeing the two last actions in a cycle, by ordering primaries (e.g., by IP address).

The full version of the algorithm (to be published separately) tolerates occasional cycles of $\rhd\!\!\!\rhd$, which require a localised consensus.

[4] An action unrelated to any other by $\rhd\!\!\!\rhd$ is considered a chain of length 0.

5 Related work

Our survey of optimistic replication [14] motivated us to understand the commonalities and differences between protocols.

The relations between consistency and ordering have been well studied in the context the causal dependence relation [12, 13]. Our simpler and modular primitives clarify and generalise this analysis. The primitives are common to all protocols, as are the significant events of actions becoming guaranteed, dead, serialised, decided and stable.

Lamport's state-machine replication [10] broadcasts actions to all sites and ensures consistency because each site executes exactly the same schedule. This is a special case of our CSMP property. Sousa et al. [17] generalise Lamport's state-machine approach to the commitment of partially replicated databases.

Much formal work on consistency focuses on serialisability [1, 5]. CSMP constitutes a generalisation of serialisability.

As seen earlier, we can encode a linearisable protocol [9] with a transition rule.

The X-Ability theory [8] allows an action to appear several times in the same schedule if it is idempotent; for instance, retrying a failed action is allowed. Schedules are tested for equivalence after filtering out such duplicates. It would be interesting to encode their approach in our formalism, and analyse their assumptions, which are quite strong. This is left for future work.

Our approach has many similarities with the Acta framework [4, 5]. Acta provides a set of logical primitives over execution histories, including presence of an event, implication, and causal dependence and ordering between events. Acta makes assumptions specific to databases, such as the existence of transaction commit and abort primitives. The Acta description language is more powerful and is used to analyze protocols at a finer granularity. On the other hand, the action-constraint language is simpler; it is straightforward to translate most of the Acta dependencies into our language. Acta takes serialisability as the definition of consistency, and does not deal with partial replication.

Constraints $\rightarrow$ and $\triangleright$ were first proposed by Fages [6] for general reconciliation problems in optimistic replication systems.

6 Conclusions and future work

We presented a formalism for describing replication protocols and consistency. Our significant subsets are common to the many replication protocols that can be described in our language. We generalise a number of classical formulations of the consistency property and prove them equivalent. This underscores the deep commonalities between protocols that appear quite different on the surface. Although consistency entails global consensus in the general case, we exhibited some sufficient conditions for making local decisions. We provided the first formal definition of consistency for partial replication, and derived a new localised algorithm for this case. Our results apply to a broad range of protocols, both

pessimistic and optimistic.

This paper only presented the intuitions; the interested reader will find a fully formal treatment in our technical report [15]. The TR also contains a detailed description for a variety of diverse classical replication protocols, including consistency proofs.

The formalism rests upon only two binary constraints. This makes it easy to prove properties, and is powerful enough to incorporate all the classical replication protocols. However the semantics of some applications (e.g., a shared bank account) demand more powerful primitives. A possible direction is to generalise constraints to be n-ary and our significant subsets to patterns. Then the crucial safety property would be that the guaranteed and dead sublanguages are disjoint.

Acknowledgments

We thank Fabrice le Fessant for his participation to early stages of this work, and Tony Hoare, Miguel Castro and Patrick Valduriez for their encouragement and suggestions.

References

[1] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987. http://research.microsoft.com/pubs/ccontrol/.

[2] A. D. Birell, R. Levin, R. M. Needham, and M. D. Schroeder. Grapevine: An exercise in distributed computing. *Commun. ACM*, 25:260–274, Apr. 1982.

[3] Y. Chong and Y. Hamadi. Distributed IceCube. Private communication, Jan. 2004.

[4] P. K. Chrysanthis and K. Ramamritham. ACTA: The SAGA continues. In A. K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, chapter 10, pages 349–397. Morgan Kaufmann, 1992.

[5] P. K. Chrysanthis and K. Ramamritham. Correctness criteria and concurrency control. In A. Sheth, A. K. Elmagarmid, and M. Rusinkiewicz, editors, *Management of Heterogeneous and Autonomous Database Systems*, chapter 10. Morgan-Kaufmann, 1998. http://www-ccs.cs.umass.edu/db/publications/mdb.ps.

[6] F. Fages. A constraint programming approach to log-based reconciliation problems for nomadic applications. In *6th Annual W. of the ERCIM Working Group on Constraints*, Prague, Czech Republic, June 2001.

[7] A. Fekete, D. Gupta, V. Luchangco, N. Lynch, and A. Shvartsman. Eventually-serializable data services. *Theoretical Computer Science*, 220(Special issue on Distributed Algorithms):113–156, 1999.

[8] S. Frølund and R. Guerraoui. X-Ability: A theory of replication. In *Symp. on Principles of Dist. Comp. (PODC 2000)*, Portland, Oregon, USA, July 2000. ACM SIGACT-SIGOPS.

[9] M. Herlihy and J. Wing. Linearizability: a correcteness condition for concurrent objects. *ACM Trans. Prog. Lang. Syst.*, 12(3):463–492, 1990.

[10] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, July 1978.

[11] N. Preguiça, M. Shapiro, and C. Matheson. Semantics-based reconciliation for collaborative and mobile environments. In *Proc. Tenth Int. Conf. on Coop. Info. Sys. (CoopIS)*, Catania, Sicily, Italy, Nov. 2003.

[12] K. Ramamritham and P. K. Chrysanthis. A taxonomy of correctness criteria in database applications. *VLDB Journal*, 5(1):85–97, 1996.

[13] M. Raynal and M. Mizuno. How to find his way in the jungle of consistency criteria for distributed shared memories (or how to escape from Minos' labyrinth). In *Proc. of the IEEE Int. Conf. on Future Trends of Distributed Computing Systems*, pages 340–346, Lisboa (Portugal), Sept. 1993.

[14] Y. Saito and M. Shapiro. Optimistic replication. Technical Report MSR-TR-2003-60, Microsoft Research, Oct. 2003. ftp://ftp.research.microsoft.com/pub/tr/tr-2003-60.pdf.

[15] M. Shapiro and K. Bhargavan. The Actions-Constraints approach to replication: Definitions and proofs. Technical Report MSR-TR-2004-14, Microsoft Research, Mar. 2004. ftp://ftp.research.microsoft.com/pub/tr/TR-2004-14.pdf.

[16] M. Shapiro, N. Preguiça, and J. O'Brien. Rufis: mobile data sharing using a generic constraint-oriented reconciler. In *Conf. on Mobile Data Management*, Berkeley, CA, USA, Jan. 2004. http://www-sor.inria.fr/~shapiro/papers/mdm-2004-final.ps.gz.

[17] A. Sousa, R. Oliveira, F. Moura, and F. Pedone. Partial replication in the database state machine. In *Int. Symp. on Network Comp. and App. (NCA'01)*, pages 298–309, Cambridge MA, USA, Oct. 2001. IEEE.

[18] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. *Trans. on Comp.-Human Interaction*, 5(1):63–108, Mar. 1998. http://doi.acm.org/10.1145/274444.274447.

[19] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing update conflicts in Bayou, a weakly connected replicated storage system. In *Proc. 15th ACM Symposium on Operating Systems Principles*, Copper Mountain CO (USA), Dec. 1995. ACM SIGOPS. http://www.acm.org/pubs/articles/proceedings/ops/224056/p172-terry/p172-terry.pdf.

[20] N. Vidot, M. Cart, J. Ferrié, and M. Suleiman. Copies convergence in a distributed real-time collaborative environment. In *Computer Supported Cooperative Work*, pages 171–180, Philadelphia, PA, USA, Dec. 2000.